Úvod do Informatiky (FI: IB000)

Doc. RNDr. Petr Hliněný, Ph.D.

hlineny@fi.muni.cz

22. listopadu 2011

Obsažný a dobře přístupný úvod do nezbytných formálních matematických základů moderní informatiky, doplněný řadou interaktivních cvičení v IS MU.

Výukový text pro předmět IB000 na FI MU od roku 2006, navazující na původní přednáškové slidy prof. Antonína Kučery z roku 2005.

Verze 1.32 (2011)

0.1 O tomto textu a jeho studiu

Vážení čtenáři,

dostává se vám do rukou výukový text Úvodu do Informatiky, který je primárně určený pro studenty stejnojmenného předmětu na FI MU Brno. Seznamuje čtenáře s několika formálními matematickými oblastmi důležitými pro úspěšné studium moderní informatiky.

Náš text svým obsahem navazuje na původní slidy předmětu IB000 sepsané A. Kučerou do roku 2005, je však výrazně rozšířený o volný text a komentáře. Mimo textu samotného (jak jej zde vidíte) jsou z téhož zdoje vytvářeny i přednáškové slidy předmětu, které najdete například v IS MU. Slidy však pochopitelně obsahují jen část textu a jsou jinak formátovány.

Učební text je psán strukturovaným matematickým přístupem, s velkým důrazem na přesnost a formalitu vyjadřování v nutných partiích. Na druhou stranu jsou strohá matematická vyjádření pokud možno doplněna obsáhlými neformálními komentáři, které mají studentům ulehčit pochopení látky. V žádném případě však čtenáři nemají zaměňovat neformální komentáře za matematické definice – v případě nejasností vždy platí to, co přesně říká formální definice.

Interaktivní osnova

`http://is.muni.cz/el/1433/podzim2011/IB000/index.qwarp`

Nedílnou součástí celého výukového textu jsou interaktivní osnova předmětu IB000 v IS MU a z ní odkazované online odpovědníky sloužící k procvičování probrané látky na jednoduchých i složitějších příkladech. To je také důvod, proč texto text obsahuje jen velmi málo příkladů k probírané látce; od každého studenta očekáváme, že si látku *bude procvičovat online na zmíněných odpovědnících*, obsahujících až tisíce příkladů desítek typů. (K praktickému pochopení přednesených znalostí i k jejich budoucím aplikacím je takové procvičení nezbytné!) V návaznosti na tyto odpovědníky doporučujeme studentům diskutovat o probírané látce a svých (ne)úspěších s cvičícími předmětu na předmětovém diskuzním fóru v IS.

Obsah

1 Základní formalismy: Důkaz a Algoritmus

Úvod

Začněme nejprve několika obecnými poznámkami ke smyslu a směřování celého našeho předmětu: Jak sami poznáte, studium informatiky neznamená jen „naučit se nějaký programovací jazyk“, nýbrž zahrnuje celý soubor dalších relevantních předmětů, mezi nimiž najdeme i matematicko–teoretické (formální) základy moderní informatiky. Právě odborný nadhled nad celou informatikou včetně nezbytné formální teorie nejspíše odliší „řadového programátora“, kterých jsou dnes spousty i bez VŠ vzdělání, od skutečného a mnohem lépe placeného počítačového experta.

A na tomto místě nyní přichází náš předmět Úvod do Informatiky, který (i přes svůj nepříliš obsažný název) vás právě na studium těchto formálních základů moderní informatiky připraví. Tak se do toho studia s chutí dejte...

Cíle

Prvním cílem této lekce je formálně rozebrat struktury matematických tvrzení (vět) a jejich formálních důkazů. V druhém sledu se naučíme obdobným způsobem formálně přesně zapisovat postupy, čili algoritmy.

1.1 Úvod do matematického dokazování

Matematika (a tudíž i teoretická informatika jako její součást) se vyznačuje velmi přísnými formálními požadavky na korektnost argumentace. Takto korektně postavená argumentace vede od přijatých předpokladů v elementárních krocích směrem k požadovanému závěru (a nikdy ne naopak!).

Definice 1.1. ***Matematický důkaz*** .
Uvažme matematickou *větu* (neboli tvrzení) tvaru

„Jestliže platí *předpoklady*, pak platí *závěr*“.

Důkaz této věty je konečná posloupnost tvrzení, kde

- každé tvrzení je buď
  - *předpoklad*, nebo
  - obecně přijatá „pravda“ – *axiom*, nebo
  - plyne z předchozích a dříve dokázaných tvrzení podle nějakého „akceptovaného“ logického principu – *odvozovacího pravidla*;
- poslední tvrzení je *závěr*.

`Komentář:` O potřebné úrovni formality matematických důkazů a o běžných důkazových technikách se dozvíme dále v této a příští lekci. Nyní si jen celou problematiku uvedeme názornými příklady.

Příklad 1.2. *Uvažujme následující matematické tvrzení (které jistě už znáte).*

Věta. *Jestliže x je součtem dvou lichých čísel, pak x je sudé.*

`Poznámka` pro připomenutí:

- *Sudé* číslo je celé číslo dělitelné 2, tj. tvaru $2k$.
- *Liché* číslo je celé číslo nedělitelné 2, tj. tvaru $2k+1$.

`Důkaz` postupuje v následujících *formálních krocích*:

	tvrzení	zdůvodnění
1)	$a = 2k+1$, k celé	předpoklad
2)	$b = 2l+1$, l celé	předpoklad
3)	$x = a+b = 2k+2l+1+1$	1,2) a komutativita sčítání (axiom)
4)	$x = 2(k+l)+2\cdot 1$	3) a distributivnost násobení
5)	$x = 2(k+l+1)$	4) a opět distributivnost násobení
6)	$x = 2m$, m celé	5) a $m = k+l+1$ je celé číslo (axiom)

□

Příklad 1.3. *Dokažte následující tvrzení:*

Věta. *Jestliže x a y jsou racionální čísla pro která platí $x < y$, pak existuje racionální číslo z pro které platí $x < z < y$.*

`Důkaz` po krocích (s již trochu méně formálním zápisem) zní:

1) Nechť $z = \frac{x+y}{2} = x + \frac{y-x}{2} = y - \frac{y-x}{2}$.
2) Číslo z je racionální, neboť x a y jsou racionální.
3) Platí $z > x$, neboť $\frac{y-x}{2} > 0$.
4) Dále platí $z < y$, neboť opět $\frac{y-x}{2} > 0$.
5) Celkem $x < z < y$.

Jak čtenář vidí, tento strohý formální zápis (i když matematicky kompletní) si zaslouží aspoň krátký vysvětlující komentář. Takový komentář, ať už se objeví před nebo hned po formálních argumentech, sice není součástí důkazu samotného, velmi však napomáhá správnému pochopení a má své nezastupitelné místo. Konkrétně je zde velmi vhodné poznamenat, že klíčový krok (1) popisuje námi vymyšlenou algebraickou konstrukci, která vede k požadovanému číslu z. Zbylé kroky (2–5) pak jen snadno zdůvodňují, že nalezené z má všechny požadované vlastnosti. □

`Poznámka`. Alternativní formulace Věty z Příkladu 1.3 mohou znít:

- Pro každé $x, y \in \mathbb{Q}$, kde $x < y$, existuje $z \in \mathbb{Q}$ takové, že $x < z < y$.
- Množina racionálních čísel je hustá.

1.2 Význam matematických vět

První krok formálního důkazu je uvědomit si, *co říká věta*, která se má dokázat; tedy co je *předpoklad* a co *závěr* dokazovaného tvrzení. *Pravdivost* takového tvrzení pak je třeba

chápat v následujícím významu:

Pro každou situaci, ve které jsou splněny všechny předpoklady,
je platný i závěr tvrzení.

`Komentář:` Příklady běžné formulace matematických *vět*:

* Konečná množina má konečně mnoho podmnožin.
* $\sin^2(\alpha) + \cos^2(\alpha) = 1$.
* Graf je rovinný, jestliže neobsahuje podrozdělení K_5 nebo $K_{3,3}$.

Co přesně nám uvedené matematické věty říkají? Často nám k lepšímu pochopení toho, co je tvrzeno a je třeba dokázat, pomůže pouhé rozepsání definic pojmů, které se v dané větě vyskytují.

O pravdivosti implikace

Všimněte si také, jaký je správný logický význam matematického tvrzení vysloveného touto formou *implikace* („jestliže ..., pak ...“). Především, pokud předpoklady nejsou splněny nebo jsou sporné, tak celé tvrzení je platné bez ohledu na pravdivost závěru!

Příklad 1.4. *Je pravdivé následující matematické tvrzení?*

> **Věta**. *Mějme dvě kuličky, červenou a modrou. Jestliže červená kulička je těžší než modrá a zároveň je modrá kulička těžší než ta červená, tak jsou obě kuličky ve skutečnosti zelené.*
>
> *„To přece nemůže být pravda, jak může být jedna kulička těžší než druhá a naopak zároveň? Jak mohou být nakonec obě zelené? To je celé nějaká blbost...“*

Ano, výše uvedené jsou typické laické reakce na uvedenou větu. Přesto však tato věta pravdivá je! Stačí se vrátit o kousek výše ke kritériu – *Pro každou situaci, ve které jsou splněny všechny předpoklady, je platný i závěr tvrzení* – které je zjevně naplněno. Nenaleznete totiž situaci, ve které by byly splněny oba předpoklady zároveň, a tudíž ve všech takových neexistujících situacích si můžete říkat cokoliv, třeba že kuličky jsou zelené. □

Příklad 1.5. *Anna a Klára přišly na přednášku a usadily se do lavic. Proč je pravdivé toto matematické tvrzení?*

> **Věta**. *Jestliže Anna sedí v první řadě lavic a zároveň Anna sedí v poslední řadě lavic, tak Klára nesedí v druhé řadě lavic.*

Opět je třeba se hluboce zamyslet nad *významem předpokladů a závěru*, ale tentkrát není situace předpokladů tak triviálně sporná, jako v Příkladu 1.4. Kdy tedy nastávají oba předpoklady zároveň? Když první řada lavic je zároveň řadou poslední! Neboli posluchárna má jen (nejvýše) jednu řadu lavic a Klára tudíž v druhé řadě nemůže sedět. Důkaz je hotov. □

1.3 Struktura matematických vět a důkazů

Jak „*moc formální*“ mají správné matematické důkazy vlastně být?

- Záleží na tom, komu je důkaz určen — *konzument* musí být schopen „snadno“ ověřit korektnost každého tvrzení v důkazu a plně pochopit, z čeho vyplývá.
- Je tedy hlavně na vás zvolit tu správnou úroveň formálnosti zápisu vět i důkazů podle situace.

A jak na ten správný matematický důkaz máme přijít?

- No…, nalézání matematických důkazů je tvůrčí činnost, která není vůbec snadná a vyžaduje od vás přímo „*umělecké*“ vlohy. Přesto se jí alespoň trochu musíte přiučit.

Dokazovat či vyvracet tvrzení?

Představme si, že našim úkolem je *rozhodnout platnost* matematického tvrzení. Jak matematicky správně zdůvodníme svou odpověď?

- Záleží na odpovědi samotné… Pokud je to ANO, prostě podáme důkaz tvrzení podle výše uvedených zvyklostí. Pokud je odpověď NE, tak naopak podáme důkaz *negace* daného tvrzení.

Poměrně častým případem pak je matematická věta T, která tvrdí nějaký závěr pro širokou oblast vstupních možností. Potom platí následující:

- Pokud T je pravdivá, nezbývá než podat *vyčerpávající důkaz* platnosti pro všechny vstupy.
- Avšak pokud T je nepravdivá, stačí „*uhodnout*“ vhodný vstupní *protipříklad* a jen pro něj dokázat, že závěr tvrzení není platný.

  `Komentář:` A nyní se znovu vraťte na začátek Oddílu 1.2 a srovnejte si výše uvedené se základními poznatky o významu matematických vět…

Příklad 1.6. *Rozhodněte platnost následujícího tvrzení: Pro všechna reálná x platí*

$$x^2 + 3x + 2 \geq 0\,.$$

`Důkaz`: Standardními algebraickými postupy si můžeme upravit vztah na $x^2+3x+2 = (x+1)\cdot(x+2) \geq 0$. Co nám z něj vyplývá? Například to, že k porušení daného tvrzení stačí volit x tak, aby jedna ze závorek byla kladná a druhá záporná. To nastane třeba pro $x = -\frac{3}{2}$.

Pro vyvrácení tvrzení nám tedy stačí začít volbou protipříkladu $x = -\frac{3}{2}$ (není nutno zdůvodňovat, jak jsme jej „uhodli“!) a následně dokázat úpravou

$$x^2 + 3x + 2 = (x+1)\cdot(x+2) = (-\frac{3}{2}+1)\cdot(-\frac{3}{2}+2) = (-\frac{1}{2})\cdot(+\frac{1}{2}) = -\frac{1}{4} < 0\,.$$

Dané tvrzení není platné. □

Konstruktivní a existenční důkazy

Z hlediska praktické využitelnosti je potřeba rozlišit tyto dvě kategorie důkazů (třebaže z formálně–matematického pohledu mezi nimi kvalitativní rozdíl není).

- Důkaz Věty 1.3 je *konstruktivní*. Dokázali jsme nejen, že číslo z existuje, ale podali jsme také návod, jak ho pro dané x a y *sestrojit*.

- *Existenční* důkaz je takový, kde se prokáže existence nějakého objektu *bez toho*, aby byl podán návod na jeho konstrukci.

Příklad 1.7. *čistě* existenčního *důkazu.*

Věta. *Existuje program, který vypíše na obrazovku čísla tažená ve 45. tahu sportky v roce 2011.*

`Důkaz`: Existuje pouze konečně mnoho možných výsledků losování 45. tahu sportky v roce 2011. Pro každý možný výsledek existuje program, který tento daný výsledek vypíše na obrazovku. Mezi těmito programy je tedy jistě ten, který vypíše právě ten výsledek, který bude ve 45. tahu sportky v roce 2011 skutečně vylosován.

□

`Komentář:` To je ale „*podvod*“, že? A přece *není*... Formálně správně to je prostě tak a tečka (2011: tedy dokud nám to pan Hušák všechno nepokazí, že?).

`Fakt`. Pro informatické i další aplikované disciplíny *je důležitá konstruktivnost* důkazů vět, které se zde objevují. V matematice ale jsou mnohé příklady užitečných a nenahraditelných existenčních důkazů, třeba tzv. *pravděpodobnostní důkazy.*

Příklad 1.8. *„pravděpodobnostního“ existenčního důkazu.*

Věta. *Na dané množině bodů je zvoleno libovolně n^2 podmnožin, každá o n prvcích. Dokažte pro dostatečně velká n, že všechny body lze obarvit dvěma barvami tak, aby žádná množina nebyla jednobarevná.*

`Důkaz`: U každého bodu si „*hodíme mincí*“ a podle výsledku zvolíme barvu červeně nebo modře. (Nezávislé volby s pravděpodobností $\frac{1}{2}$.) Pravděpodobnost, že zvolených n bodů vyjde jednobarevných, je jen $\frac{2}{2^n} = 2^{1-n}$.

Pro n^2 podmnožin tak je pravděpodobnost, že některá z nich vyjde jednobarevná, shora ohraničená součtem

$$\underbrace{2^{1-n} + \cdots + 2^{1-n}}_{n^2} = \frac{n^2}{2^{n-1}} \to 0\,.$$

Jelikož je toto číslo (pravděpodobnost) pro $n \geq 7$ menší než 1, bude existovat obarvení bez jednobarevných zvolených podmnožin. □

1.4 Formální popis algoritmu

Nakonec si položme otázku, co je vlastně algoritmus? Když se na tím zamyslíte, asi zjistíte, že to není tak jednoduché přesně říci. Dokonce je to natolik obtížná otázka, že si zde podáme jen zjednodušenou odpověď.

`Poznámka`: Za definici algoritmu je obecně přijímána tzv. *Church–Turingova teze* tvrdící, že všechny algoritmy lze „simulovat“ na Turingově stroji. Jedná se sice o přesnou, ale značně nepraktickou definici. Mimo Turingova stroje existují i jiné *matematické modely výpočtů*, jako třeba stroj RAM, který je abstrakcí skutečného strojového kódu, nebo také třeba Definice 9.2 v našem textu.

Konvence 1.9. Zjednodušeně zde *algoritmem* rozumíme konečnou posloupnost elementárních výpočetních *kroků*, ve které každý další krok vhodně[1] využívá (neboli závisí na) vstupní údaje či hodnoty vypočtené v předchozích krocích. Tuto závislost přitom pojímáme zcela obecně nejen na operandy, ale i na vykonávané instrukce v krocích.

Pro zápis algoritmu a jeho zpřehlednění a zkrácení využíváme *řídící konstrukce* – podmíněná větvení a cykly.

`Komentář:` Vidíte, jak blízké si jsou konstruktivní matematické důkazy a algoritmy v našem pojetí? Jedná se nakonec o jeden ze záměrů našeho přístupu...

Příklad 1.10. *Zápis algoritmu pro výpočet průměru z daného pole* `a[]` *s* `n` *prvky.*

Algoritmus.

- Inicializujeme `sum` ← `0` ;
- postupně pro `i=0,1,2,...,n-1` provedeme
  - `sum` ← `sum+a[i]` ;
- vypíšeme podíl `(sum/n)` . □

Ve „vyšší úrovni“ formálnosti (s jasnějším vyznačením *řídících struktur* algoritmu) se totéž dá zapsat jako:

Algoritmus 1.11. ***Průměr***
z daného pole `a[]` *s* `n` *prvky.*

```
sum ← 0;
foreach  i ← 0,1,2,...,n-1  do
    sum ← sum+a[i];
done
res ← sum/n;
output res;
```

Symbolický zápis algoritmů

`Značení`. Pro potřeby symbolického formálního zápisu algoritmů v předmětu FI: IB000 si zavedeme následovnou konvenci:

- *Proměnné* nebudeme deklarovat ani typovat, pole odlišíme závorkami `p[]`.
- *Přiřazení* hodnoty zapisujeme `a` ← `b`, případně `a := b`, ale nikdy ne `a=b`.
- Jako elementární operace je možné použít jakékoliv *aritmetické výrazy* v běžném matematickém zápise. Rozsahem a přesností čísel se zde nezabýváme.
- Podmíněné *větvení* uvedeme klíčovými slovy `if ... then ... else ... fi` , kde `else` větev lze vynechat (a někdy, na jednom řádku, i `fi`).
- Pevný *cyklus* uvedeme klíčovými slovy `foreach ... do ... done` , kde část za `foreach` musí obsahovat *předem danou* množinu hodnot pro přiřazování do řídící proměnné.

[1] Zvídaví studenti si mohou na tomto místě uvědomit, že ve slůvku „vhodně“ se skrývá celá hloubka Church–Turingovy teze. V žádném případě tak nelze uvedené obracet, že by každá posloupnost kroků atd ... byla algoritmem ve smyslu této teze (viz také Lekce 11).

* *Podmíněný cyklus* uvedeme klíčovými slovy `while ... do ... done`. Zde se může za `while` vyskytovat jakákoliv logická podmínka.

* V zápise používáme jasné *odsazování* (zleva) podle úrovně zanoření řídících struktur (což jsou `if, foreach, while`).

* Pokud je to dostatečně jasné, elementární operace nebo podmínky můžeme i ve formálním zápise *popsat běžným jazykem*.

Malé srovnání závěrem

Jak to tedy je s vhodnou a únosnou mírou formalizace u matematických důkazů i u zápisu algoritmů? Odpověď může naznačit tato tabulka:

	zcela formální	*běžná úroveň*
`matematika`	formální rozepsání všech elem. kroků (Příklad 1.2)	strukturovaný a matem. přesný text v běžném jazyce
`algoritmy`	rozepsání všech elem. kroků ve výpočetním modelu	strukturovaný rozpis kroků (Algoritmus 1.11), i s využitím běžného jazyka
`programování`	assembler / strojový kód (kde se s ním dnes běžně setkáte?)	„vyšší“ (strukturované) programovací jazyky, například Java

`Komentář:` Pochopitelně se ve všech třech bodech obvykle držíme druhého přístupu, tedy běžné úrovně formality, pokud si specifické podmínky výslovně nevyžadují přístup nejvyšší formality...

Navazující studium

Látka této úvodní lekce má velmi široký záběr. S potřebou formálního zápisu tvrzení a důkazů se setkáte nejen v tomto, ale asi i ve všech dalších matematických předmětech, které zároveň studujete či budete studovat, a principy budou stále stejné. Taktéž schopnost formálně rozepisovat a správně chápat algoritmy je nezbytná pro celé příští studium informatiky. Blíže se formálním zápisům algoritmů a jejich dokazování budeme věnovat i v Lekcích 8,9,10.

2 Důkazové techniky, Indukce

Úvod

Náš hlubší úvod do matematických formalismů pro informatiku pokračujeme základním přehledem technik matematických důkazů. Třebaže matematické dokazování a příslušné techniky mohou někomu připadat neprůhledné (a možná zbytečné), jejich pochopení a zvládnutí není samoúčelné, neboť nám pomáhá si mnoho uvědomit o studovaných problémech samotných. Konečně, jak si můžeme být jisti svými poznatky, když bychom pro ně nebyli schopni poskytnout důkazy?

Během studia tohoto předmětu poznáte (a ti méně šťastní až s překvapením u zkoušek), že vlastně vše, k čemu naším výkladem směřujeme, se dá neformálně shrnout slovy „naučit se přesně vyjadřovat a být si svými tvrzeními naprosto jisti“ a analogicky „naučit se navrhovat správné algoritmy a být si i svými programy naprosto jisti“.

Z důkazových postupů je pro nás informatiky asi nejdůležitější technika důkazů matematickou indukcí, *která je svou podstatou velmi blízká počítačovým programům (coby iterace cyklů). V dalším výkladu budeme indukci hojně využívat, především v Lekcích 8 a 10.*

Cíle

Cílem této lekce je popsat základní techniky matematických důkazů, z nichž největší důraz klademe na matematickou indukci. Každý student by se měl alespoň naučit formálně psané důkazy číst a pochopit. (Pro vysvětlení, z Lekce 1 víme, že matematický důkaz má „jednotnou formu“ Definice 1.1, ale nyní se věnujeme takříkajíc šablonám, podle nichž takový korektní důkaz sestavíme.)

2.1 Přehled základních důkazových technik

V matematice je často používaných několik následujících způsobů – technik, jak k danému tvrzení nalézt korektní formální důkaz. (Uvědomme si, že jedno tvrzení mívá mnoho různých, stejně korektních, důkazů; ty se však mohou výrazně lišit svou složitostí.) Tyto techniky si v bodech shrneme zde:

- *Přímé odvození.* To je způsob, o kterém jsme se dosud bavili.
  Postupujeme přímo od předpokladů k závěru, ale sami poznáte, že taková „přímá“ cesta je obtížně k nalezení.

- *Kontrapozice* (také *obrácením* či *nepřímý důkaz*). Místo věty

  > „Jestliže platí *předpoklady*, pak platí *závěr*.“

  budeme dokazovat ekvivalentní větu

  > „Jestliže neplatí *závěr*, pak neplatí alespoň jeden z *předpokladů*.“

- *Důkaz sporem.* Místo věty

  > „Jestliže platí *předpoklady*, pak platí *závěr*.“

  budeme dokazovat větu

  > „Jestliže platí *předpoklady* a platí *opak závěru*, pak platí opak jednoho z *předpokladů* (nebo platí jiné *zjevně nepravdivé tvrzení*).“

- *Matematická indukce.* Pokročilá technika, kterou zde popíšeme později...

Tyto techniky jsou asi nejlépe ilustrovány následovnými příklady důkazů.

Příklad důkazu kontrapozicí

`Definice`: *Prvočíslo $p > 1$ nemá jiné dělitele než 1 a p.*

Příklad 2.1. *na důkaz kontrapozicí (obrácením).*

Věta. *Jestliže p je prvočíslo větší než 2, pak p je liché.*

`Důkaz`: *Obráceného tvrzení* – budeme tedy dokazovat, že

je-li p sudé, pak p buď není větší než 2, nebo p není prvočíslo.

Připomínáme, že podle definice je p sudé, právě když lze psát $p = 2 \cdot k$, kde k je celé. Jsou dvě možnosti:

- $k \leq 1$. Pak $p = 2k$ není větší než 2.
- $k > 1$. Pak $p = 2 \cdot k$ není prvočíslo podle definice. □

`Poznámka`: Důkazy kontrapozicí pracují s *negací* (opakem) *předpokladů* a *závěru*. Je-li např. *závěr* komplikované tvrzení tvaru

„z toho, že z A a B plyne C, vyplývá, že z A nebo C plyne A a B“,

není pouhou intuicí snadné zjistit, co je vlastně jeho negací. Jak uvidíme v pozdějších lekcích, užitím jednoduché induktivní metody lze podobná tvrzení negovat zcela *mechanicky*.

Příklady důkazu sporem

Příklad 2.2. *Jiný přístup k Důkazu 2.1.*

Věta. *Jestliže p je prvočíslo větší než 2, pak p je liché.*

`Důkaz` *sporem*: Nechť tedy p je prvočíslo větší než 2, které je sudé. Pak $p = 2 \cdot k$ pro nějaké $k > 1$, tedy p není prvočíslo, spor (s předpokladem, že p je prvočíslo). □

Důkaz *sporem* je natolik specifický a důležitý v matematice, že si zaslouží širší vysvětlení. Co je vlastně jeho podstatou? Je to (zcela přirozený) předpoklad, že v *konzistentní teorii* nelze zároveň odvodit tvrzení i jeho negaci. Jestliže tedy ve schématu

„Jestliže platí *předpoklady* a platí *opak závěru*, pak platí opak jednoho z *předpokladů*, nebo platí jiné *zjevně nepravdivé tvrzení*.“

odvodíme k některému předpokladu jeho spor, nebo případně jiné tvrzení, které odporuje všeobecně přijatým faktům (například $0 = 1$), pak něco musí být „špatně“. Co však v našem tvrzení může (nezapomeňte předpoklad konzistence) být chybné? Původní předpoklady byly dány, takže zbývá jedině náš dodatečný předpoklad, že platí *opak závěru*. Tudíž opak závěru nemůže nikdy platit a dvojí negací odvodíme, že platí původní závěr.

Příklad 2.3.

Věta. *Číslo $\sqrt{2}$ není racionální.*

`Důkaz` *sporem*: Nechť tedy $\sqrt{2}$ je racionální, tj. nechť existují nesoudělná celá kladná čísla r, s taková, že $\sqrt{2} = r/s$.

– Pak $2 = r^2/s^2$, tedy $r^2 = 2 \cdot s^2$, proto r^2 je dělitelné dvěma. Z toho plyne, že i r je dělitelné dvěma (proč?).

– Jelikož r je dělitelné dvěma, je r^2 dělitelné dokonce čtyřmi, tedy $r^2 = 4 \cdot m$ pro nějaké m. Pak ale také $4 \cdot m = 2 \cdot s^2$, tedy $2 \cdot m = s^2$ a proto s^2 je dělitelné dvěma.

– Z toho plyne, že s je také dělitelné dvěma. Celkem dostáváme, že r i s jsou dělitelné dvěma, jsou tedy soudělná a to je spor (s definicí racionálního čísla). □

`Komentář:` „Nevíte-li, jak nějakou větu dokázat, zkuste důkaz sporem...“

2.2 Věty typu „tehdy a jen tehdy“

Uvažujme nyní (v matematice poměrně hojné) *věty* tvaru

> „Nechť platí *předpoklady P*. Pak *tvrzení A* platí *tehdy a jen tehdy*, platí-li *tvrzení B*.“

Příklady jiných formulací téže věty jsou:

* Za předpokladů P je tvrzení B *nutnou a postačující* podmínkou pro platnost tvrzení A.
* Za předpokladů P je tvrzení A nutnou a postačující podmínkou pro platnost tvrzení B.
* Nechť platí předpoklady P. Pak tvrzení A platí *právě tehdy* když platí tvrzení B.

`Fakt`: *Důkaz* vět tohoto tvaru má vždy *dvě části(!)*. Je třeba dokázat:

* Jestliže platí předpoklady P a tvrzení A, pak platí tvrzení B.
* Jestliže platí předpoklady P a tvrzení B, pak platí tvrzení A.

2.3 Matematická indukce

Pokud se souhrnně podíváme na důkazové techniky v matematice, všimneme si, že matematická indukce je skoro „dvorní“ důkazovou technikou diskrétní matematiky. To proto, že umožňuje pohodlně dokazovat i složitá tvrzení po jednotlivých (diskrétních) krocích od počátečního.

Uvažme tedy větu ve tvaru:

> „*Pro každé přirozené (celé)* $n \geq k_0$ *platí* $T(n)$.“

Zde k_0 je nějaké pevné přir. číslo a $T(n)$ je tvrzení parametrizované čís. n. Příkladem je třeba tvrzení:

> *Pro každé* $n \geq 0$ *platí, že* n *přímek dělí rovinu nejvýše na* $\frac{1}{2}n(n+1)+1$ *oblastí.*

Definice 2.4. Princip matematické indukce říká, že k důkazu věty

> „*Pro každé přirozené (celé)* $n \geq k_0$ *platí* $T(n)$.“

stačí ověřit platnost těchto dvou tvrzení:

- $T(k_0)$ (tzv. *báze* neboli základ indukce)

- *Pro každé $n \geq k_0$; jestliže platí $T(n)$,* (indukční předpoklad)
  pak platí také $T(n+1)$. (indukční krok)

Formálně řečeno, matematická indukce je axiomem aritmetiky přirozených čísel.

`Poznámka`: Pozor, v tomto předmětu počítáme 0 za přirozené číslo!

Opět jako v předešlém si tuto techniku ilustrujeme množstvím názorných příkladů.

Příklady důkazů indukcí

Příklad 2.5. *Velmi jednoduchá a přímočará indukce.*

Věta. *Pro každé $n \geq 1$ je stejná pravděpodobnost, že při současném hodu n kostkami bude výsledný součet sudý, jako, že bude lichý.*

`Důkaz`: *Základ indukce* je zde zřejmý: Na jedné kostce (poctivé!) jsou tři lichá a tři sudá čísla, takže obě skupiny padají se stejnou pravděpodobností.

Indukční krok pro $n \geq 1$: Nechť p_n^s pravděpodobnost, že při hodu n kostkami bude výsledný součet sudý, a p_n^l je pravděpodobnost lichého. Podle indukčního předpokladu je $p_n^s = p_n^l = \frac{1}{2}$. Hoďme navíc $(n+1)$-ní kostkou. Podle toho, zda na ní padne liché nebo sudé číslo, je pravděpodobnost celkového sudého součtu rovna

$$\frac{3}{6}p_n^l \;+\; \frac{3}{6}p_n^s \;=\; \frac{1}{2}$$

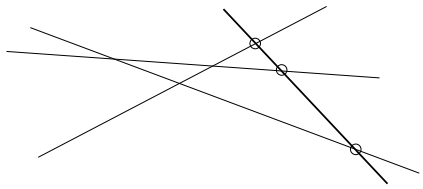
a stejně pro pravděpodobnost celkového lichého součtu. □

Příklad 2.6. *Ukázka důkazové „síly“ principu matematické indukce.*

Věta. *Pro každé $n \geq 0$ platí, že n přímek dělí rovinu nejvýše na*

$$\frac{1}{2}n(n+1)+1$$

oblastí.



`Důkaz`: Pro bázi indukce stačí, že 0 přímek dělí rovinu na jednu část. (Všimněte si také, že 1 přímka dělí rovinu na dvě části, jen pro lepší pochopení důkazu.)

Mějme nyní rovinu rozdělenou n přímkami na nejvýše $\frac{1}{2}n(n+1)+1$ částí. Další, $(n+1)$-ní přímka je rozdělena průsečíky s předchozími přímkami na nejvýše $n+1$ úseků a každý z nich oddělí novou část roviny. Celkem tedy bude rovina rozdělena našimi přímkami na nejvýše tento počet oblastí:

$$\frac{1}{2}n(n+1)+1 \;+\; (n+1) \;=\; \frac{1}{2}n(n+1) + \frac{1}{2}\cdot 2(n+1) \;+\; 1 \;=\; \frac{1}{2}(n+1)(n+2) \;+\; 1$$

□

Příklad 2.7. *Další indukční důkaz rozepsaný v podrobných krocích.*

Věta. *Pro každé* $n \geq 0$ *platí* $\sum_{j=0}^{n} j = \frac{n(n+1)}{2}$.

`Důkaz` *indukcí* vzhledem k n.

- *Báze:* Musíme dokázat tvrzení $T(0)$, což je v tomto případě rovnost $\sum_{j=0}^{0} j = \frac{0(0+1)}{2}$. Tato rovnost (zjevně) platí.

- *Indukční krok:* Musíme dokázat následující tvrzení:

$$\textit{Jestliže platí } \sum_{j=0}^{i} j = \tfrac{i(i+1)}{2} \textit{ kde } i \geq 0, \textit{ pak platí } \sum_{j=0}^{i+1} j = \tfrac{(i+1)(i+2)}{2}.$$

Předpokládejme tedy, že $\sum_{j=0}^{i} j = \frac{i(i+1)}{2}$ a pokusme se dokázat, že pak také $\sum_{j=0}^{i+1} j = \frac{(i+1)(i+1+1)}{2} = \frac{(i+1)(i+2)}{2}$. To už plyne úpravou:

$$\sum_{j=0}^{i+1} j \;=\; \sum_{j=0}^{i} j \;+ (i+1) \;=\; \frac{i(i+1)}{2} + (i+1) \;=\; \frac{i(i+1)+2(i+1)}{2} \;=\; \frac{(i+1)(i+2)}{2}$$

Podle principu matematické indukce je celý důkaz hotov. □

`Poznámka`: Výsledek Příkladu 2.7 je ukázkou tzv. sumačního vzorce pro posloupnost přirozených čísel. Jinou ukázkou je třeba vztah

$$1 + 3 + \cdots + (2n-3) + (2n-1) = 1 + (2n-1) \;+\; 3 + (2n-3) \;+ \cdots = 2n \cdot \frac{n}{2} = n^2,$$

nebo $1^2 + 2^2 + \cdots + n^2 = \frac{1}{6}n(n+1)(2n+1)$ a $1^3 + 2^3 + \cdots + n^3 = (1+2+\cdots+n)^2 = \frac{1}{4}n^2(n+1)^2$. Odvozování a práce s jednoduchými sumačními vzorci by také měla náležet do výbavy schopného informatika, princip matematické indukce je zde klíčový.

2.4 Komentáře k matematické indukci

Pro správné a úspěšné použití indukce v dokazování je vhodné si zapamatovat několik cenných rad:

* Základní trik všech důkazů matematickou indukcí je vhodná *reformulace* tvrzení $T(n+1)$ tak, aby se „odvolávalo“ na tvrzení $T(n)$.
  - Dobře se vždy podívejte, v čem se liší tvrzení $T(n+1)$ od tvrzení $T(n)$. Tento „rozdíl“ budete muset v důkaze zdůvodnit.

* Pozor, občas je potřeba „*zesílit*“ tvrzení $T(n)$, aby indukční krok správně „fungoval“.
  - Velkým problémem bohužel je, že není možno podat návod, jak vhodné „zesílení“ nalézt (ani kdy jej vůbec hledat). Jedná se vlastně o pokusy a „hádání z křišťálové koule“.

* Často se chybuje v důkazu indukčního kroku, neboť ten bývá většinou výrazně obtížnější než báze, ale o to zrádnější jsou chyby v samotné zdánlivě snadné bázi!
  - Dejte si dobrý pozor, od které hodnoty $n \geq k_0$ je indukční krok univerzálně platný...

Zesílení indukčního kroku

Příklad 2.8. *Když je nutno indukční krok zesílit…*

Věta. *Pro každé $n \geq 1$ platí*

$$s(n) = \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \cdots + \frac{1}{n(n+1)} < 1\,.$$

Důkaz: *Báze indukce* je zřejmá, neboť $\frac{1}{1 \cdot 2} < 1$.
Co však *indukční krok*? Předpoklad $s(n) < 1$ je sám o sobě „*příliš slabý*“ na to, aby bylo možno tvrdit $s(n+1) = s(n) + \frac{1}{(n+1)(n+2)} < 1$.

Neznamená to ještě, že by tvrzení nebylo platné, jen je potřeba náš indukční předpoklad *zesílit*. Budeme dokazovat

Tvrzení. *Pro každé přirozené $n \geq 1$ platí $s(n) \leq 1 - \frac{1}{n+1} < 1$.*

To platí pro $n = 1$ (nezapomeňte ověřit!) a dále už úpravou jen dokončíme zesílený indukční krok:

$$s(n+1) \;=\; s(n) + \frac{1}{(n+1)(n+2)} \;\leq\; 1 - \frac{1}{n+1} + \frac{1}{(n+1)(n+2)} =$$

$$= \; 1 + \frac{-(n+2)+1}{(n+1)(n+2)} \;=\; 1 - \frac{1}{n+2} \qquad \square$$

Rozšíření báze a předpokladu

Mimo zesilování tvrzení indukčního kroku jsme někdy okolnostmi nuceni i k *rozšiřování* samotné báze indukce a s ní indukčního předpokladu na více než jednu hodnotu parametru n.

- Můžeme například předpokládat platnost (parametrizovaných) tvrzení $T(n)$ *i* $T(n+1)$ *zároveň*, a pak odvozovat platnost $T(n+2)$. Toto lze samozřejmě zobecnit na jakýkoliv počet předpokládaných parametrů.

- Můžeme dokonce předpokládat platnost tvrzení $T(j)$ *pro všechna* $j = k_0, k_0+1, \ldots, n$ najednou a dokazovat $T(n+1)$. Toto typicky využijeme v případech, kdy indukční krok „rozdělí“ problém $T(n+1)$ na dvě menší části a z nich pak odvodí platnost $T(n+1)$.

Fakt: Obě prezentovaná „rozšíření“ jsou v konečném důsledku jen speciálními instancemi základní matematické indukce; použité rozšířené možnosti pouze zjednodušují formální zápis důkazu.

Příklad 2.9. *Když je nutno rozšířit bázi a indukční předpoklad…*

Věta. *Nechť funkce f pro každé $n \geq 0$ splňuje vztah $f(n+2) = 2f(n+1) - f(n)$. Pokud platí $f(0) = 1$ a zároveň $f(1) = 2$, tak platí $f(n) = n + 1$ pro všechna přirozená $n \geq 0$.*

Důkaz: Už samotný pohled na daný vztah $f(n+2) = 2f(n+1) - f(n)$ naznačuje, že bychom měli *rozšířit indukční předpoklad* (a krok) zhruba takto:

Pro každé $n \geq 0$; jestliže platí $T(n)$; neboli $f(n) = n+1$, a zároveň platí $T(n+1)$; $f(n+1) = n+2$, pak platí také $T(n+2)$; $f(n+2) = n+3$.

Báze indukce – pozor, zde už musíme ověřit *dvě* hodnoty

$$f(0) = 0 + 1 = 1, \quad f(1) = 1 + 1 = 2\,.$$

Náš *indukční krok* tak nyní může využít celého rozšířeného předpokladu, znalosti hodnot $f(n)$ i $f(n+1)$, pro ověření

$$f(n+2) = 2f(n+1) - f(n) = 2 \cdot (n+1+1) - (n+1) = n+3 = n+2+1\,.$$

□

`Komentář:` Jak by tento důkaz měl být formulován v „tradiční“ indukci? („Substitucí“ nového tvrzení.)

Závěrem malý „problém“

Příklad 2.10. *Aneb jak snadno lze v matematické indukci udělat chybu.*

Věta. *(„nevěta“)*
V každém stádu o $n \geq 1$ koních mají všichni koně stejnou barvu.

`Důkaz` indukcí vzhledem k n.
Báze: Ve stádu o jednom koni mají všichni koně stejnou barvu.
Indukční krok: Nechť $S = \{K_1, \ldots, K_{n+1}\}$ je stádo o $n+1$ koních. Dokážeme, že všichni koně mají stejnou barvu. Uvažme dvě menší stáda:

– $S' = \{K_1, K_2, \ldots, K_n\}$

– $S'' = \{K_2, \ldots, K_n, K_{n+1}\}$

Podle indukčního předpokladu mají všichni koně ve stádu S' stejnou barvu B'. Podobně všichni koně ve stádu S'' mají podle indukčního předpokladu stejnou barvu B''. Dokážeme, že $B' = B''$, tedy že všichni koně ve stádu S mají stejnou barvu. To ale plyne z toho, že koně $K_2, \ldots, K_n$ patří jak do stáda S', tak i do stáda S''. □

`Komentář:` *Ale to už je podvod! Vidíte, kde?*

Navazující studium

Jak jsme již řekli, matematické důkazy a jejich chápání jsou nezbytné ke studiu vysokoškolských matematických předmětů. Bez schopnosti přesného vyjadřování a chápání definic a vět se informatik neobejde, ani pokud se zaměřuje čistě aplikovaným směrem; viz třeba správné pochopení různých norem a specifikací.

Na druhou stranu umění "tvořit" nové matematické důkazy je dosti obtížné a nedá se jemu jen tak snadno naučit – vyžaduje to mnoho pokročilé praxe. Jelikož je schopnost formálního matematického dokazování nezbytná (převážně jen) v teoretických informatických disciplínách, není tato část kritická v celém předmětu (a u zkoušek se objeví s menším důrazem), ale to neznamená, že byste se jí mohli zcela vyhýbat. Obzvláště techniku matematické indukce by měl každý informatik aspoň trochu ovládat, neboť s jejím použitím se zajisté ještě mnohokráte setkáte v budoucím studiu.

3 Množiny a jejich Prvky

Úvod

V přehledu matematických formalismů informatiky se v této lekci zaměříme na základní datové typy matematiky, tj. na množiny, relace a funkce. Netradiční sousloví „datové typy" matematiky zde volíme záměrně, abychom zdůraznili jejich fundamentálnost pro výstavbu navazující teorie v analogii k programování.

O množinách jste sice zajisté slyšeli už na základní škole, ale podstatou našeho předmětu je uvést povětšinou neformálně známé pojmy na patřičnou formální úroveň nutnou pro teoretické základy informatiky. V případě konečné teorie množin si zde vystačíme s tzv. „naivním" pohledem, který dostatečně matematicky přesně vystihuje všechny jejich konečné aspekty (o aspektech nekonečných množin se krátce zmíníme až v Lekci 11).

Cíle

V této lekci si ukážeme první krok k seriózně vybudované matematické teorii množin – tzv. naivní teorii množin, která nám velmi dobře poslouží ve všech konečných případech. Na to navážeme zavedením základního množinového kalkulu, shrnutím běžných vlastností množin a uvedeme si příklady rekurzivně definovaných posloupností.

3.1 Pojem množiny

Položme si hned na úvod tu nejdůležitější otázku: Co je vlastně ***množina***?
Na tuto otázku bohužel není zcela jednoduchá odpověď... Abychom se vůbec někam v našem úvodu dostali, spokojíme se zatím jen s přirozeným „naivním pohledem".

`Definice` naivní teorie množin: „*Množina je soubor prvků a je svými prvky plně určena.*"

`Komentář:` Příklady zápisu množin výčtem prvků
$\emptyset$, $\{a,b\}$, $\{b,a\}$, $\{a,b,a\}$, $\{\{a,b\}\}$, $\{\emptyset,\{\emptyset\},\{\{\emptyset\}\}\}$, $\{x \mid x \text{ je liché přirozené číslo}\}$.

Pozor, pojem *prvku* sám o sobě nemá matematický význam, svého významu totiž nabývá pouze ve spojení „*být prvkem množiny*". Prvky množiny tak může být cokoliv, mimo jiné i dalším množiny.

`Komentář:` Relativitu významu vztahu „prvek–množina" si můžeme přiblížit třeba na vztahu „*podřízený–nadřízený*" z běžného pracovního života. Tam také nemá smysl jen říkat, že je někdo podřízeným, aniž řekneme také jeho nadřízeného. Přitom i vedoucí je někomu ještě podřízený a naopak i ten poslední podřízený pracovník může být pánem třeba svého psa. Podobně je tomu s množinou jako „*nadřízenou*" svých prvků.

`Značení` množin a jejich prvků:

- $x \in M$ „x je *prvkem* množiny M",
- $\emptyset$ je *prázdná* množina $\{\}$.

`Komentář:` Některé vlastnosti vztahu „být prvkem" jsou
- $a \in \{a,b\}$, $a \notin \{\{a,b\}\}$, $\{a,b\} \in \{\{a,b\}\}$, $a \notin \emptyset$, $\emptyset \in \{\emptyset\}$, $\emptyset \notin \emptyset$,
- *rovnost* množin dle svých prvků $\{a,b\} = \{b,a\} = \{a,b,a\}$, $\{a,b\} \neq \{\{a,b\}\}$.

`Značení`: Počet prvků (*mohutnost*) množiny A zapisujeme $|A|$.

`Komentář:` $|\emptyset| = 0$, $|\{\emptyset\}| = 1$, $|\{a,b,c\}| = 3$, $|\{\{a,b\},c\}| = 2$.

Jednoduché srovnání množin

Vztah „*být prvkem množiny*“ přirozeně nám podává i způsob porovnávání množin mezi sebou. Jedná se o klíčovou část teorie množin.

`Definice`: Množina A je *podmnožinou* množiny B, právě když každý prvek A je prvkem B. Píšeme $A \subseteq B$ nebo obráceně $B \supseteq A$. Říkáme také, že se jedná o *inkluzi*.

* Platí $\{a\} \subseteq \{a\} \subseteq \{a, b\} \not\subseteq \{\{a, b\}\}, \quad \emptyset \subseteq \{\emptyset\}$,
* $A \subsetneq B$ právě když $A \subseteq B$ a $A \neq B$ (A je *vlastní* podmnožinou B).

Z naivní definice množiny pak přímo vyplývá následující:

`Definice`: Dvě množiny jsou si *rovny* $A = B$ právě když $A \subseteq B$ a $B \subseteq A$.

* Podle naivní definice jsou totiž množiny A a B stejné, mají-li stejné prvky.
* Důkaz rovnosti množin $A = B$ má obvykle *dvě části*: Odděleně se dokáží inkluze $A \subseteq B$ a $B \subseteq A$.

Ukázky nekonečných množin

`Značení`: Běžné číselné množiny v matematice jsou následující

* $\mathbb{N} = \{0, 1, 2, 3, \dots\}$ je množina přirozených čísel,
* $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, 3, \dots\}$ je množina celých čísel,
* $\mathbb{Z}^+ = \{1, 2, 3, \dots\}$ je množina celých kladných čísel,
* $\mathbb{Q}$ je množina racionálních čísel (zlomků).
* $\mathbb{R}$ je množina reálných čísel.

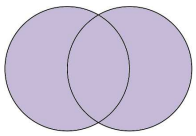
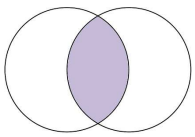
`Komentář:` Tyto uvedené číselné množiny jsou vesměs *nekonečné*, na rozdíl od konečných množin uvažovaných v předchozím „naivním“ pohledu. Pojem nekonečné množiny se přímo v matematice objevil až teprve v 19. století a bylo s ním spojeno několik *paradoxů* ukazujících, že naivní pohled na teorii množin pro nekonečné množiny *nedostačuje*. My se k problematice nekonečných množin, Kantorově větě a Russelově paradoxu vrátíme v závěru našeho předmětu v Lekci 11.

3.2 Množinové operace

Začněme se základními operacemi množinového kalkulu, které zajisté na intuitivní úrovni již ovládáte ze školy. Pro formální úplnost podáme jejich přesné definice, na které se můžete odvolávat v důkazech.

Definice 3.1. ***Sjednocení*** $\cup$ ***a průnik*** $\cap$ dvou množin A, B definujeme

$$\begin{aligned} A \cup B &= \{x \mid x \in A \text{ nebo } x \in B\}, \\ A \cap B &= \{x \mid x \in A \text{ a současně } x \in B\}. \end{aligned}$$

$A \cup B$  $A \cap B$ 

* Příklady $\{a,b,c\} \cup \{a,d\} = \{a,b,c,d\}$, $\{a,b,c\} \cap \{a,d\} = \{a\}$.
* Vždy platí „distributivita“ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ a $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

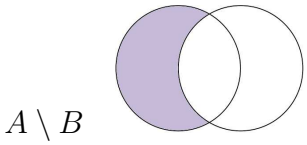
`Definice`: Pro libovolný počet množin indexovaných pomocí I rozšířeně definujeme

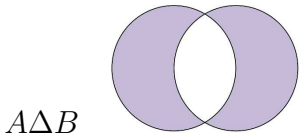
$$\begin{aligned} \bigcup_{i\in I} A_i &= \{x \mid x \in A_i \text{ pro } \textit{nějaké } i \in I\}, \\ \bigcap_{i\in I} A_i &= \{x \mid x \in A_i \text{ pro } \textit{každé } i \in I\}. \end{aligned}$$

`Komentář:` Nechť $A_i = \{2 \cdot i\}$ pro každé $i \in \mathbb{N}$. Pak $\bigcup_{i\in\mathbb{N}} A_i$ je množina všech sudých přirozených čísel. Nechť $B_i = \{x \mid x \in \mathbb{N}, x \geq i\}$ pro každé $i \in \mathbb{N}$. Pak $\bigcap_{i\in\mathbb{N}} B_i = \emptyset$.

Definice 3.2. ***Rozdíl*** $\setminus$ ***a symetrický rozdíl*** Δ dvou množin A, B definujeme

$$\begin{aligned} A \setminus B &= \{x \mid x \in A \text{ a současně } x \notin B\}, \\ A \,\Delta\, B &= \big(A \setminus B\big) \cup \big(B \setminus A\big). \end{aligned}$$





* Příklady $\{a,b,c\} \setminus \{a,b,d\} = \{c\}$, $\{a,b,c\}\Delta\{a,b,d\} = \{c,d\}$.
* Vždy platí například $A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C)$ apod.

`Definice`: Nechť $A \subseteq M$. *Doplněk A vzhledem k M* je množina $\overline{A} = M \setminus A$.

`Komentář:` Jedná se o poněkud specifickou operaci, která *musí být vztažena* vzhledem k *nosné množině M*! Je-li $M = \{a,b,c\}$, pak $\overline{\{a,b\}} = \{c\}$. Je-li $M = \{a,b\}$, pak $\overline{\{a,b\}} = \emptyset$.

* Vždy pro $A \subseteq M$ platí $\overline{\overline{A}} = A$ („dvojí“ doplněk).
* Vždy pro $A, B \subseteq M$ platí $\overline{A \cup B} = \overline{A} \cap \overline{B}$ a $\overline{A \cap B} = \overline{A} \cup \overline{B}$. (Viz Vennovy diagramy.)

Uspořádané dvojice a kartézský součin

Zatímco prvky v množinách jsou zcela *neuspořádané*, jsou mnohé situace, kdy musíme pracovat se „*seřazenými*“ výčty prvků. V teorii množin lze takovéto seřazení definovat oklikou následovně:

`Definice`: *Uspořádaná dvojice* (a,b) je zadána množinou $\{\{a\},\{a,b\}\}$.

`Fakt`: Platí $(a,b) = (c,d)$ právě když $a = c$ a současně $b = d$. Uměli byste toto sami dokázat přímo z podané definice?

* Co je podle definice (a,a)? Je to $(a,a) = \{\{a\},\{a,a\}\} = \{\{a\},\{a\}\} = \{\{a\}\}$.

Definice 3.3. ***Kartézský součin*** dvou množin A, B definujeme jako množinu všech uspořádaných dvojic ze složek z A a B

$$A \times B = \{(a, b) \mid a \in A, b \in B\}\,.$$

* Příklady $\{a, b\} \times \{a\} = \{(a, a), (b, a)\}$, $\{c, d\} \times \{a, b\} = \{(c, a), (c, b), (d, a), (d, b)\}$.
* Platí $\emptyset \times X = \emptyset = X \times \emptyset$ pro každou množinu X.
* Jednoduchá mnemotechnická pomůcka říká $|A \times B| = |A| \cdot |B|$.

`Definice`: Pro každé $k \in \mathbb{N}, k > 0$ definujeme *uspořádanou k-tici* $(a_1, \cdots, a_k)$ induktivně
- $(a_1) = a_1$,
- $(a_1, \cdots, a_i, a_{i+1}) = ((a_1, \cdots, a_i), a_{i+1})$.

Všimněte si, že definice používá tzv. rekurentní vztah, blíže viz Oddíl 3.4.

`Fakt`: Platí $(a_1, \cdots, a_k) = (b_1, \cdots, b_k)$ právě když $a_i = b_i$ pro každé $i \in \mathbb{N}$ kde $1 \leq i \leq k$.

`Definice` *kartézského součinu* více množin: Pro každé $k \in \mathbb{N}$ definujeme

$$A_1 \times \cdots \times A_k = \{(a_1, \cdots, a_k) \mid a_i \in A_i \text{ pro každé } 1 \leq i \leq k\}\,.$$

* Například $\mathbb{Z}^3 = \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} = \{(i, j, k) \mid i, j, k \in \mathbb{Z}\}$.
* Co je A^0? $\{\emptyset\}$, neboť jediná uspořádaná 0-tice je právě prázdná $\emptyset$.

`Poznámka`: Podle uvedené definice *není součin asociativní*, tj. obecně nemusí platit, že $A \times (B \times C) = (A \times B) \times C$. V matematické praxi je někdy výhodnější uvažovat upravenou definici, podle níž součin *asociativní je*. Pro účely této přednášky není podstatné, k jaké definici se přikloníme. Prezentované definice a věty „fungují“ pro obě varianty.

Potenční množina

Definice 3.4. ***Potenční množina*** množiny A,
neboli množina všech podmnožin, je definovaná vztahem

$$2^A = \{B \mid B \subseteq A\}\,.$$

* Platí například $2^{\{a,b\}} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$,
* $2^\emptyset = \{\emptyset\}$, $\quad 2^{\{\emptyset,\{\emptyset\}\}} = \{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \{\emptyset, \{\emptyset\}\}\}$,
* $2^{\{a\}\times\{a,b\}} = \{\emptyset, \{(a, a)\}, \{(a, b)\}, \{(a, a), (a, b)\}\}$.

Věta 3.5. *Počet prvků potenční množiny splňuje* $|2^A| = 2^{|A|}$.

`Důkaz`: Stručně indukcí podle $|A|$: Pro $A = \emptyset$ platí $|2^A| = |\{\emptyset\}| = 1$. Pro každý další prvek $b \notin A$ rozdělíme všechny podmnožiny $A \cup \{b\}$ „napolovic“ na ty neobsahující b a na ty obsahující b, tudíž

$$\left|2^{A\cup\{b\}}\right| = 2 \cdot \left|2^A\right| = 2^{|A|+1} = 2^{|A\cup\{b\}|}\,. \qquad \square$$

3.3 Porovnávání a určení množin

Pro obecnou ilustraci formálního množinového kalkulu si ukažme dva důkazy rovností mezi množinovými výrazy. Podobně lze (rozepsáním příslušných definic) rutinně dokazovat další množinové vztahy.

Věta 3.6. *Pro každé dvě množiny $A, B \subseteq M$ platí $\overline{A \cup B} = \overline{A} \cap \overline{B}$.*

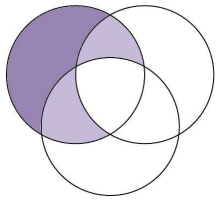
`Důkaz` v obou směrech rovnosti.

- $\overline{A \cup B} \subseteq \overline{A} \cap \overline{B}$: Pro $x \in M$ platí $x \in \overline{A \cup B}$, právě když $x \notin A \cup B$, neboli když zároveň $x \notin A$ a $x \notin B$. To znamená $x \in \overline{A}$ a zároveň $x \in \overline{B}$, z čehož vyplývá požadované $x \in \overline{A} \cap \overline{B}$.
- $\overline{A \cup B} \supseteq \overline{A} \cap \overline{B}$: Pro $x \in M$ platí $x \in \overline{A} \cap \overline{B}$, právě když $x \in \overline{A}$ a zároveň $x \in \overline{B}$, neboli když zároveň $x \notin A$ a $x \notin B$. To znamená $x \notin A \cup B$, z čehož vyplývá požadované $x \in \overline{A \cup B}$.

□

Věta 3.7. *Pro každé tři množiny A, B, C platí*

$$A \setminus (B \cap C) \;=\; (A \setminus B) \;\cup (A \setminus C)\,.$$



`Důkaz` (viz ilustrační obrázek).

- $A \setminus (B \cap C) \;\subseteq\; (A \setminus B) \;\cup (A \setminus C)$: Je-li $x \in A \setminus (B \cap C)$, pak $x \in A$ a zároveň $x \notin (B \cap C)$, neboli $x \notin B$ nebo $x \notin C$. Pro první možnost máme $x \in (A \setminus B)$, pro druhou $x \in (A \setminus C)$.
- Naopak $A \setminus (B \cap C) \;\supseteq\; (A \setminus B) \;\cup (A \setminus C)$: Je-li $x \in (A \setminus B) \;\cup (A \setminus C)$, pak $x \in (A \setminus B)$ nebo $x \in (A \setminus C)$. Pro první možnost máme $x \in A$ a zároveň $x \notin B$, z čehož plyne $x \in A$ a zároveň $x \notin (B \cap C)$, a tudíž $x \in A \setminus (B \cap C)$. Druhá možnost je analogická.

□

Charakteristický vektor (pod)množiny

V případech, kdy všechny uvažované množiny jsou podmnožinami nějaké konečné *nosné množiny* X, což není neobvyklé v programátorských aplikacích, s výhodou využijeme následující reprezentaci množin.

`Definice`: Mějme nosnou množinu $X = \{x_1, x_2, \ldots, x_n\}$. Pro $A \subseteq X$ definujeme *charakteristický vektor* χ_A jako

$$\chi_A = (c_1, c_2, \ldots, c_n), \text{ kde } c_i = 1 \text{ pro } x_i \in A \text{ a } c_i = 0 \text{ jinak.}$$

Užitečnost této reprezentace je ilustrována také následovnými fakty.

- Platí $A = B$ právě když $\chi_A = \chi_B$.
- Množinové operace jsou realizovány „bitovými funkcemi“
  sjednocení $\sim$ OR, průnik $\sim$ AND, symetrický rozdíl $\sim$ XOR.

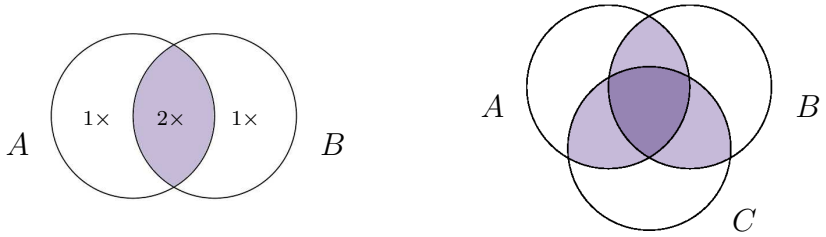
Princip inkluze a exkluze

Tento důležitý a zajímavý kombinatorický princip je někdy také nazýván „princip *zapojení a vypojení*“. Používá se ke zjišťování počtů prvků množin s neprázdnými průniky.

Věta 3.8. *Počet prvků ve sjednocení dvou či tří množin spočítáme:*

$$|A \cup B| = |A| + |B| - |A \cap B|$$

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$



`Důkaz`: Uvedeme si podrobný důkaz prvního vztahu $|A \cup B| = |A| + |B| - |A \cap B|$, přičemž druhá část je analogická (viz obrázková ilustrace). Nechť $x \in A \cup B$. Pokud $x \notin B$, tak je $x \in A$ a prvek x je započítán na pravé straně $|A| + |B| - |A \cap B|$ právě jednou v $|A|$. Obdobně je to v případě $x \notin A$. Nakonec pro $x \in A \cap B$ je ve vztahu $|A| + |B| - |A \cap B|$ tento prvek x započítán také $1 + 1 - 1 = 1$ krát. □

`Komentář:` Všimněte si, že Větu 3.8 lze stejně tak využít k výpočtu počtu prvků v průniku množin...

Příklad 3.9. *Z* 1000 *televizí jich při první kontrole na výrobní lince má* 5 *vadnou obrazovku,* 10 *je poškrábaných a* 12 *má jinou vadu. Přitom* 3 *televize mají současně všechny tři vady a* 4 *jiné jsou poškrábané a mají jinou vadu. Kolik televizí je celkem vadných?*

Řešení: Dosazením $|A| = 5$, $|B| = 10$, $|C| = 12$, $|A \cap B \cap C| = 3$, (zde pozor) $|A \cap B| = 3 + 0$, $|A \cap C| = 3 + 0$, $|B \cap C| = 3 + 4$ do Věty 3.8 zjistíme výsledek 17. □

`Poznámka`. Jen stručně, bez důkazu a bližšího vysvětlení, si uvedeme *obecnou formu principu inkluze a exkluze*:

$$\left| \bigcup_{j=1}^{n} A_j \right| = \sum_{\emptyset \neq I \subseteq \{1,\ldots,n\}} (-1)^{|I|-1} \cdot \left| \bigcap_{i \in I} A_i \right|$$

(Jeho znalost nebude v předmětu vyžadována.)

3.4 Posloupnosti a rekurentní vztahy

Uspořádané k-tice (Oddíl 3.2) jsou také nazývány *konečnými posloupnostmi* délky k, neboli seřazeními prvků z dané neprázdné množiny hodnot (opakování prvků je povoleno). Tento pohled lze přirozeně zobecnit na nekonečné posloupnosti, avšak musíme sáhnout k trochu pokročilejším pojmům (viz také Oddíl 4.1 o funkcích):

`Definice`: *Nekonečná posloupnost* p je funkcí z $\mathbb{N}$ do svého oboru hodnot H, neboli $p : \mathbb{N} \to H$. Mimo „funkčního“ zápisu $p(n)$ často používáme „indexovou“ formu zápisu funkční hodnoty p_n.

`Poznámka`: Oborem hodnot posloupnosti obvykle bývá nějaká číselná množina, ale může to být i jakákoliv jiná množina. Také definiční obor posloupnosti může začínat od nuly nebo i od jedničky, jak je v aplikacích potřeba.

- Příklady posloupností:
  - $p_0 = 0$, $p_1 = 2, \ldots, p_i = 2i, \ldots$ je posloupnost sudých nezáporných čísel.
  - 3, 3.1, 3.14, 3.141, ... je posloupnost postupných dekadických rozvojů π.
  - $1, -1, 1, -1, \ldots$ je posloupnost určená vztahem $p_i = (-1)^i$, $i \geq 0$.
  - Pokud chceme stejnou posloupnost $1, -1, 1, -1, \ldots$ zadat jako q_i, $i \geq 1$, tak ji určíme vzorcem $q_i = (-1)^{i-1}$.
- Posloupnost je *rostoucí* (či *klesající*), pokud $p_{n+1} > p_n$ ($p_{n+1} < p_n$) pro všechna n.

Rekurentní definice posloupnosti

Slovem *rekurentní* označujeme takové definice (či popisy), které se v jistých bodech odvolávají samy na sebe. (Už jste se setkali s „rekurzí“ při programování? A víte, co znamená?) Místo nepřehledných formálních definic si *rekurentní vztahy* uvedeme několika názornými ukázkami.

- Zadáme-li posloupnost p_n vztahy $p_0 = 1$ a $p_n = 2p_{n-1}$ pro $n > 0$, pak platí $p_n = 2^n$ pro všechna n.
- Obdobně můžeme zadat posloupnost q_n vztahy $q_1 = 1$ a $q_n = q_{n-1} + n$ pro $n > 1$. Potom platí $q_n = \frac{1}{2}n(n+1)$ pro všechna n. Uměli byste toto dokázat indukcí? Viz Příklad 2.7.
- Známá Fibonacciho posloupnost je zadaná vztahy $f_1 = f_2 = 1$ a $f_n = f_{n-1} + f_{n-2}$ pro $n > 2$.

Příklad 3.10. *Posloupnost f je zadaná rekurentní definicí*

$$f(0) = 3 \quad \text{a} \quad f(n+1) = 2 \cdot f(n) + 1$$

pro všechna přirozená n. Určete hodnotu $f(n)$ explicitním vzorcem v závislosti na n.

Řešení: V první fázi řešení takového příkladu musíme nějak „uhodnout“ hledaný vzorec pro $f(n)$. Jak? Zkusíme vypočítat několik prvních hodnot a uvidíme...

$$\begin{aligned} f(1) &= 2 \cdot f(0) + 1 = 2 \cdot 3 + 1 = 7 \\ f(2) &= 2 \cdot f(1) + 1 = 2 \cdot 7 + 1 = 15 \\ f(3) &= 2 \cdot f(2) + 1 = 2 \cdot 15 + 1 = 31 \\ f(4) &= 2 \cdot f(3) + 1 = 2 \cdot 31 + 1 = 63 \end{aligned}$$

Nepřipomínají nám tato čísla něco? Co třeba posloupnost $8-1, 16-1, 32-1, 64-1\ldots$? Bystrému čtenáři se již asi podařilo uhodnout, že půjde o mocniny dvou snížené o 1. Přesněji, $f(n) = 2^{n+2} - 1$ (proč je exponent $n+2$? inu proto, aby správně vyšlo $f(0)$).

Ve druhé nesmíme ale zapomenout správnost našeho „věštění“ dokázat, nejlépe matematickou indukcí podle n.

- Báze: $f(0) = 2^{0+2} - 1 = 4 - 1 = 3$ platí.

- Indukční krok: za využití indukčního předpokladu

$$f(n+1) = 2 \cdot f(n) + 1 = 2 \cdot (2^{n+2} - 1) + 1 = 2 \cdot 2^{n+2} - 2 + 1 = 2^{(n+1)+2} - 1,$$

  což také platí.

Podle principu matematické indukce je nyní dokázáno, že pro zadanou rekurentní posloupnost f platí $f(n) = 2^{n+2} - 1$ pro všechna přirozená n. □

Navazující studium

I když jste se s "množinami" setkávali už od základní školy, do matematické teorie množin asi nahlédnete na VŠ poprvé. Nezalekněte se na úvod formality vyjadřování, která je bohužel nezbytná, a naučte se s množinami a relacemi dobře pracovat aspoň na zde ukázané naivní úrovni konečné teorie množin. (Lehký náhled na obecně nekonečné množiny a jejich zdánlivé paradoxy i informatické aplikace si ukážeme později v Lekci 11...)

Dále si srovnejte pojem posloupnosti s pojmem funkce v Oddílu 4.1 a také si později povšimněte blízké koncepční podobnosti rekurentních definic s induktivními definicemi množin a funkcí v Oddíle 6.5.

4 Relace a funkce, Ekvivalence

Úvod

V návaznosti na předchozí lekci si podrobně rozebereme matematické formalismy relací a funkcí. Na rozdíl od množin, které se v jisté velmi naivní formě objevují už na základní škole, se relacím *v jejich abstraktní podobě moc pozornosti ve výuce nevěnuje. Stejně tak s pojmem* funkce *jste se již setkali na nižších stupních škol, ale povětšinou jen ve spojení s aritmetickými a analytickými funkcemi tvaru* $x + 1$, $x^2 - y$, *či* $\sin x$, $1 + \cos x^2$, *atd.*

Jak uvidíte nyní, pojmy relace i funkce jsou zcela abstraktní a neváže se ně žádný analytický vzorec výpočtu či podobné. Přitom na pojem (zcela obecné) relace velmi brzo narazí každý informatik při studiu dat a databází. Není to však jen oblast relačních databází, ale i jiná místa informatiky, kde se obecné funkce a relace skrývají či přímo explicitně objevují. Nejčastěji se setkáte s binárními relacemi, například vždy, když rozdělujete objekty podle „shodných" znaků (relace ekvivalence*), nebo když objekty mezi sebou „porovnáváte" (relace* uspořádání*). Na tyto dvě základní oblasti se dále zaměříme.*

Cíle

Úkolem této lekce je vybudovat matematický aparát (konečných) relací, s primárním zaměřením na funkce a binární relace typu ekvivalence a uspořádání. Ve vztahu k binárním relacím je zavedeno množství pojmů, které jsou později užitečné v různých oblastech matematiky i informatiky. Látka pak plynule pokračuje Lekcí 5.

4.1 Relace a funkce nad množinami

Vedle množin dalším důležitým základním „*datovým typem*" matematiky jsou relace, kterým vzhledem k jejich rozsáhlému použití v informatice věnujeme významnou pozornost v této i dvou příštích lekcích.

Definice 4.1. ***Relace*** mezi množinami $A_1, \cdots, A_k$, pro $k \in \mathbb{N}$,
je *libovolná* podmnožina kartézského součinu

$$R \subseteq A_1 \times \cdots \times A_k \,.$$

Pokud $A_1 = \cdots = A_k = A$, hovoříme o *k-ární relaci R na A*.

`Komentář:` Příklady relací.

* $\{(1, a), (2, a)\}$ je relace mezi $\{1, 2, 3\}$ a $\{a, b\}$.
* $\{(i, 2 \cdot i) \mid i \in \mathbb{N}\}$ je *binární* relace na $\mathbb{N}$.
* $\{(i, j, i + j) \mid i, j \in \mathbb{N}\}$ je *ternární* relace na $\mathbb{N}$.
* $\{3 \cdot i \mid i \in \mathbb{N}\}$ je *unární* relace na $\mathbb{N}$.
* Jaký význam vlastně mají unární a nulární relace na A?

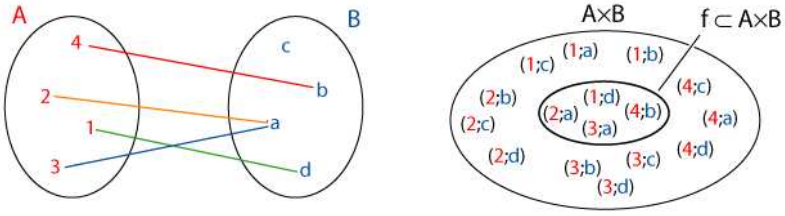
Uvědomme si, jak obecně je relace definována – její definice umožňuje podchytit skutečně libovolné „vztahy" mezi prvky téže i různých množin. V praxi se relace velmi široce využívají třeba v relačních databázích...

Funkce mezi množinami

Za první setkání s instancemi relací se většinou dá považovat pojem funkce. Přitom tradiční „školní“ pojetí *funkce* ji obvykle identifikuje s nějakým výpočetním (analytickým) předpisem či vzorcem. Pojem funkce je však daleko obecnější a zcela abstraktní.

Definice 4.2. ***(Totální) funkce*** z množiny A do množiny B
je relace f mezi A a B taková, že pro každé $x \in A$ existuje právě jedno $y \in B$ takové, že $(x, y) \in f$. Množina A se nazývá *definiční obor* a množina B *obor hodnot* funkce f.

`Komentář:` Neformálně řečeno, ve funkci f je každé „vstupní“ hodnotě x přiřazena jednoznačně „výstupní“ hodnota y. (V obecné relaci počty „přiřazených“ dvojic neomezujeme...)



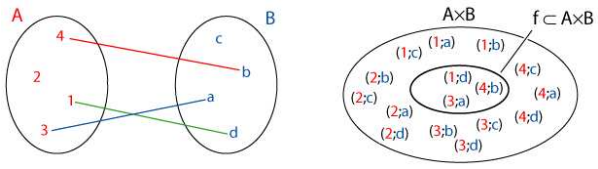
`Značení`: Místo $(x, y) \in f$ píšeme obvykle $f(x) = y$. Zápis $f : A \to B$ říká, že f je funkce s definičním oborem A a oborem hodnot B. Funkcím se také říká *zobrazení*.

`Komentář:` Příklady funkcí jsou třeba následující.

* Definujeme funkci $f : \mathbb{N} \to \mathbb{N}$ předpisem $f(x) = x + 8$. Pak $f = \{(x, x + 8) \mid x \in \mathbb{N}\}$.
* Definujeme funkci $plus : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ předpisem $plus(i, j) = i + j$.
  Pak $plus = \{(i, j, i + j) \mid i, j \in \mathbb{N}\}$.

`Definice`: Pokud naši Definici 4.2 upravíme tak, že požadujeme pro každé $x \in A$ nejvýše jedno $y \in B$ takové, že $(x, y) \in f$, obdržíme definici *parciální funkce* z A do B.

`Komentář:`



V parciální funkci f nemusí být pro některé „vstupní“ hodnoty x funkční hodnota definována (viz například $f(2)$ v uvedeném obrázku).

Pro *nedefinovanou* hodnotu používáme znak $\perp$.

`Komentář:` Následuje několik příkladů parciálních funkcí.

* Definujeme parciální funkci $f : \mathbb{Z} \to \mathbb{N}$ předpisem

$$f(x) = \begin{cases} 3 + x & \text{jestliže } x \geq 0, \\ \perp & \text{jinak.} \end{cases}$$

  Tj. $f = \{(x, 3 + x) \mid x \in \mathbb{N}\}$.
* Také funkce $f : \mathbb{R} \to \mathbb{R}$ daná běžným analytickým předpisem $f(x) = \log x$ je jen parciální – není definována pro $x \leq 0$.
* Co je relace, přiřazující lidem v ČR jejich (česká) rodná čísla?

4.2 Reprezentace konečných relací

Oblastí, kde informatici nejčastěji potkají obecné relace, je bezesporu ukládání dat. To proto, že shromažďovaná data, stejně jako relace, především sledují vztahy mezi objekty.

Příklad 4.3. *Tabulky relační databáze.*

Definujme následující množiny („elementární typy“)

* ZNAK $= \{a, \cdots, z, A, \cdots, Z, mezera\}$,
* CISLICE $= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Dále definujeme tyto množiny („odvozené typy“)

* JMENO $=$ ZNAK^{15}, PRIJMENI $=$ ZNAK^{20},
* VEK $=$ CISLICE^3,
* ZAMESTNANEC $\sim$ JMENO $\times$ PRIJMENI $\times$ VEK.

Relaci „typu“ *ZAMESTNANEC* pak lze reprezentovat tabulkou:

JMENO	*PRIJMENI*	*VEK*
Jan	Novák	42
Petr	Vichr	28
Pavel	Zíma	26
Stanislav	Novotný	52

□

`Poznámka`: *Relační databáze* je konečná množina tabulek. *Schéma databáze* je (zjednodušeně řečeno) množina „typů“ jednotlivých tabulek.

Reprezentace binárních relací na množině

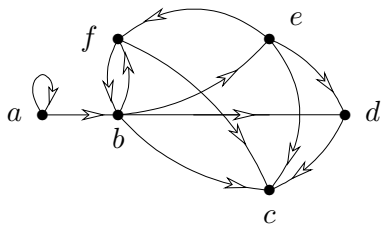
Jistě čtenáři uznají, že zadání relace výčtem jejích složek není pro člověka (na rozdíl od počítače) tím nejpříjemnějším způsobem. Je tedy přirozené se ptát, jak co nejnázorněji takovou relaci, alespoň v její nejčastější binární podobě, ukázat.

`Značení`: Binární relaci $R \subseteq M \times M$ lze jednoznačně znázornit jejím *grafem*:

- Prvky M znázorníme jako body v rovině.
- Prvek $(a, b) \in R$ znázorníme jako *orientovanou hranu* („šipku“) z a do b. Je-li $a = b$, pak je touto hranou „smyčka“ na a.

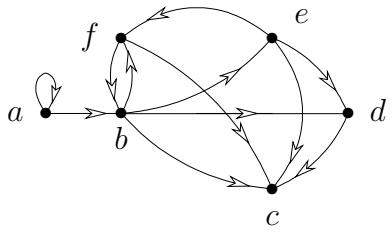
`Komentář:` Pozor, nejedná se o „grafy funkcí“ známé z matematické analýzy.

Například mějme $M = \{a, b, c, d, e, f\}$ a $R = \{(a, a), (a, b), (b, c), (b, d), (b, e), (b, f), (d, c), (e, c), (f, c), (e, d), (e, f), (f, b)\}$, pak:



V případě, že M je nekonečná nebo „velká“, může být reprezentace R jejím grafem nepraktická (záleží také na míře „pravidelnosti“ R).

`Značení`: Binární relaci $R \subseteq M \times M$ lze jednoznačně zapsat také pomocí *matice* relace – matice $\boldsymbol{A}$ typu $M \times M$ s hodnotami z $\{0, 1\}$, kde $a_{i,j} = 1$ právě když $(i, j) \in R$.



$$\rightarrow \quad \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} = \boldsymbol{A}$$

4.3 Vlastnosti binárních relací

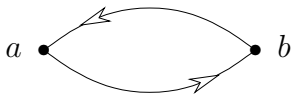
Pro začátek si výčtem a doprovodnými schematickými obrázky uvedeme pět základních vlastností binárních relací, které nás typicky v matematice zajímají.

Definice 4.4. Nechť $R \subseteq M \times M$. Binární relace R je

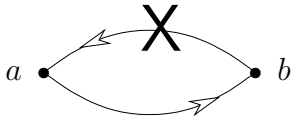
- *reflexivní*, právě když pro každé $a \in M$ platí $(a, a) \in R$;

- *ireflexivní*, právě když pro každé $a \in M$ platí $(a, a) \notin R$;

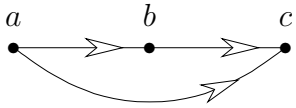
- *symetrická*, právě když pro každé $a, b \in M$ platí, že jestliže $(a, b) \in R$, pak také $(b, a) \in R$;



- *antisymetrická*, právě když pro každé $a, b \in M$ platí, že jestliže $(a, b), (b, a) \in R$, pak $a = b$;



- *tranzitivní*, právě když pro každé $a, b, c \in M$ platí, že jestliže $(a, b), (b, c) \in R$, pak také $(a, c) \in R$.

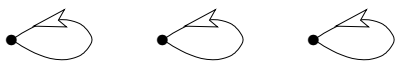


Následují dva základní typy binárních relací; kde R je

- relace *ekvivalence*, právě když je R reflexivní, symetrická a tranzitivní;
- *částečné uspořádání*, právě když je R reflexivní, antisymetrická a tranzitivní (často říkáme jen *uspořádání*).

`Komentář:` Pozor, může být relace *symetrická i antisymetrická zároveň*? Ano!

Vezměte si relaci $R = \{(x, x) \mid x \in M\}$, která obě jmenované vlastnosti splňuje. Proto pokud jste třeba dotázáni, zda relace je symetrická, nestačí se v odpovědi odvolávat na fakt, že je antisymetrická(!), neboť tyto vlastnosti se nevylučují.



Příklad 4.5. *Několik příkladů relací definovaných v přirozeném jazyce.*

Nechť M je množina všech studentů 1. ročníku FI. Uvažme postupně relace $R \subseteq M \times M$ definované takto

* $(x, y) \in R$ právě když x a y mají stejné rodné číslo;
* $(x, y) \in R$ právě když x má stejnou výšku jako y (dejme tomu na celé mm);
* $(x, y) \in R$ právě když výška x a y se neliší více jak o 2 mm;
* $(x, y) \in R$ právě když x má alespoň takovou výšku jako y;
* $(x, y) \in R$ právě když x má jinou výšku než y (dejme tomu na celé mm);
* $(x, y) \in R$ právě když x je zamilován(a) do y.

Zamyslete se podrobně, které z definovaných vlastností tyto jednotlivé relace mají. Které z nich tedy jsou ekvivalencí nebo uspořádáním? □

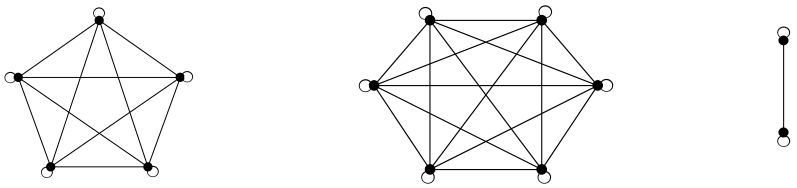
Příklad 4.6. *Jaké vlastnosti mají následující relace?*

* Buď $R \subseteq \mathbb{N} \times \mathbb{N}$ definovaná takto $(x, y) \in R$ právě když x dělí y. (*Částečné uspořádání*, ale ne každá dvě čísla jsou porovnatelná.)
* Buď $R \subseteq \mathbb{N} \times \mathbb{N}$ definovaná takto $(x, y) \in R$ právě když x a y mají stejný zbytek po dělení číslem 5. (*Ekvivalence.*)
* Nechť $F = \{f \mid f : \mathbb{N} \to \mathbb{N}\}$ je množina funkcí. Buď $R \subseteq F \times F$ definovaná takto $(f, g) \in R$ právě když $f(x) < g(x)$ pro všechna x. (*Ireflexivní*, *antisymetrická* a *tranzitivní*, ale ne reflexivní – striktně řečeno není uspořádáním.) □

4.4 Relace ekvivalence

Podle Definice 4.4 je relace $R \subseteq M \times M$ *ekvivalence* právě když R je reflexivní, symetrická a tranzitivní. Tyto *tři vlastnosti* tedy musí být splněny a ověřeny k důkazu toho, že daná relace R je ekvivalence.

`Komentář:` Jak vypadá *graf relace ekvivalence*? Poměrně příznačně, jak nám ukazuje následující obrázek (všimněte si absence šipek, která je dána symetrií relace).



Neformálně řečeno; ekvivalence je relace $R \subseteq M \times M$, taková, že $(x, y) \in R$ právě když x a y jsou v nějakém smyslu „*stejné*“.

`Značení`. V případě relace ekvivalence se někdy lze setkat s pojmenováním jako $\sim$ či $\approx$ místo R. Místo $(x, y) \in R$ se pak píše $x \approx y$.

Příklad 4.7. *Nechť M je množina všech studentů 1. ročníku FI. Uvažme postupně relace $R \subseteq M \times M$ definované následovně a zkoumejme, zda se jedná o ekvivalence:*

* $(x, y) \in R$ právě když x má stejnou výšku jako y;
* $(x, y) \in R$ právě když x má stejnou barvu vlasů jako y;
* $(x, y) \in R$ právě když x, y mají stejnou výšku a stejnou barvu vlasů;
* $(x, y) \in R$ právě když x, y mají stejnou výšku *nebo* stejnou barvu vlasů.

U kterého body se nejedná o relaci ekvivalence a proč? Je to poslední případ, kdy není splněna tranzitivita. □

Uvedený příklad ukazuje na náaledující univerzální poznatek, který může platit (a taky platí) pouze pro průnik a nikoliv pro sjednocení.

Tvrzení 4.8. *Nechť R, S jsou dvě relace ekvivalence na stejné množině M. Pak jejich* průnik *$R \cap S$ je opět relací ekvivalence.*

`Důkaz` (náznak): Jelikož reflexivita a symetrie je zřejmá, stačí snadno ověřit platnost tranzitivity na $R \cap S$. Nechť $(a, b), (b, c) \in R \cap S$, pak podle tranzitivity každé samostatné z R, S plyne $(a, c) \in R$ a zároveň $(a, c) \in S$. □

Příklad 4.9. *Nechť $R \subseteq \mathbb{N} \times \mathbb{N}$ je binární relace definovaná takto: $(x, y) \in R$ právě když $|x - y|$ je dělitelné třemi.*

V jakém smyslu jsou zde x a y „stejné“? Dávají stejný zbytek po dělení třemi. □

Příklad 4.10. *Buď R binární relace mezi všemi studenty na přednášce FI:IB000 definovaná takto: $(x, y) \in R$ právě když x i y sedí v první lavici.*

Už na „první pohled“ jde o relaci symetrickou a tranzitivní. Proč se v tomto případě *nejedná* o relaci ekvivalence? Protože není reflexivní pro studenty sedící v dalších lavicích. (Takže si dávejte dobrý pozor na správné pochopení definic.) □

4.5 Rozklady a jejich vztah k ekvivalencím

Náplní následující části výkladu je ukázat jiný přirozený pohled na ekvivalence. Tento nový pohled nám matematicky formalizuje představu ekvivalence jako rozdělení prvků nosné množiny M na „*hromádky*“, přičemž prvky každé jednotlivé hromádky jsou si navzájem v jistém smyslu „stejné“.

Definice 4.11. ***Rozklad množiny.*** Nechť M je množina.
Rozklad (na) M je množina podmnožin $\mathcal{N} \subseteq 2^M$ splňující následující tři podmínky:
- $\emptyset \notin \mathcal{N}$ (tj. každý prvek $\mathcal{N}$ je neprázdná podmnožina M);
- pokud $A, B \in \mathcal{N}$, pak buď $A = B$ nebo $A \cap B = \emptyset$;
- $\bigcup_{A \in \mathcal{N}} A = M$.

Prvkům $\mathcal{N}$ se také říká *třídy rozkladu*.

`Komentář:`
* Buď $M = \{a, b, c, d\}$. Pak $\mathcal{N} = \{\{a\}, \{b, c\}, \{d\}\}$ je rozklad na M.
* Nechť $A_0 = \{k \in \mathbb{N} \mid k \bmod 3 = 0\}$, $A_1 = \{k \in \mathbb{N} \mid k \bmod 3 = 1\}$, $A_2 = \{k \in \mathbb{N} \mid k \bmod 3 = 2\}$. Pak $\mathcal{N} = \{A_0, A_1, A_2\}$ je rozklad všech přirozených čísel $\mathbb{N}$ podle zbytkových tříd.

Každý rozklad $\mathcal{N}$ na M jednoznačně určuje jistou ekvivalenci $R_{\mathcal{N}}$ na M:

Věta 4.12. *Nechť M je množina a $\mathcal{N}$ rozklad na M. Nechť $R_{\mathcal{N}} \subseteq M \times M$ je relace na M definovaná takto*

$$(x, y) \in R_{\mathcal{N}} \text{ právě když existuje } A \in \mathcal{N} \text{ taková, že } x, y \in A.$$

Pak $R_{\mathcal{N}}$ je ekvivalence *na M.*

`Důkaz`: Dokážeme, že $R_{\mathcal{N}}$ je *reflexivní, symetrická a tranzitivní* (*Definice 4.4*).

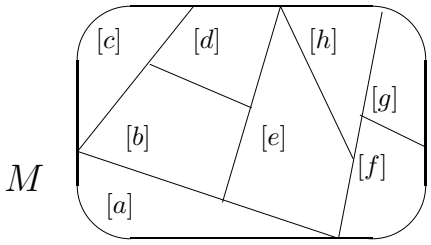
- Reflexivita: Buď $x \in M$ libovolné. Jelikož $\mathcal{N}$ je rozklad na M, musí existovat $A \in \mathcal{N}$ takové, že $x \in A$ (jinak spor se třetí podmínkou z *Definice 4.11*). Proto $(x, x) \in R_{\mathcal{N}}$, tedy $R_{\mathcal{N}}$ je reflexivní.
- Symetrie: Nechť $(x, y) \in R_{\mathcal{N}}$. Podle definice $R_{\mathcal{N}}$ pak existuje $A \in \mathcal{N}$ taková, že $x, y \in A$. To ale znamená, že také $(y, x) \in R_{\mathcal{N}}$ podle definice $R_{\mathcal{N}}$, tedy $R_{\mathcal{N}}$ je symetrická.
- Tranzitivita: Nechť $(x, y), (y, z) \in R_{\mathcal{N}}$. Podle definice $R_{\mathcal{N}}$ existují $A, B \in \mathcal{N}$ takové, že $x, y \in A$ a $y, z \in B$. Jelikož $A \cap B \neq \emptyset$, podle druhé podmínky z *Definice 4.11* platí $A = B$. Tedy $x, z \in A = B$, proto $(x, z) \in R_{\mathcal{N}}$ podle definice $R_{\mathcal{N}}$. □

Každá ekvivalence R na M naopak jednoznačně určuje jistý rozklad M/R na M:

Věta 4.13. *Nechť M je množina a R ekvivalence na M. Pro každé $x \in M$ definujeme množinu*

$$[x] = \{y \in M \mid (x, y) \in R\}.$$

Pak $\{[x] \mid x \in M\}$ je rozklad *na M, který značíme M/R a čteme „rozklad M podle R“.*



`Důkaz`: Dokážeme, že M/R splňuje podmínky *Definice 4.11*.

- Pro každé $[x] \in M/R$ platí $[x] \neq \emptyset$, neboť $x \in [x]$.
- Nechť $[x], [y] \in M/R$. Ukážeme, že pokud $[x] \cap [y] \neq \emptyset$, pak $[x] = [y]$.

  Jestliže $[x] \cap [y] \neq \emptyset$, existuje $z \in M$ takové, že $z \in [x]$ a $z \in [y]$. Podle definice $[x]$ a $[y]$ to znamená, že $(x, z), (y, z) \in R$. Jelikož R je symetrická a $(y, z) \in R$, platí $(z, y) \in R$. Jelikož $(x, z), (z, y) \in R$ a R je tranzitivní, platí $(x, y) \in R$. Proto také $(y, x) \in R$ (opět ze symetrie R). Nyní dokážeme, že $[y] = [x]$:

  - „$[x] \subseteq [y]$“: Nechť $v \in [x]$. Pak $(x, v) \in R$ podle definice $[x]$. Dále $(y, x) \in R$ (viz výše), tedy $(y, v) \in R$ neboť R je tranzitivní. To podle definice $[y]$ znamená, že $v \in [y]$.
  - „$[y] \subseteq [x]$“: Nechť $v \in [y]$. Pak $(y, v) \in R$ podle definice $[y]$. Dále $(x, y) \in R$ (viz výše), tedy $(x, v) \in R$ neboť R je tranzitivní. To podle definice $[x]$ znamená, že $v \in [x]$.

- Platí $\bigcup_{[x] \in M/R} [x] = M$, neboť $x \in [x]$ pro každé $x \in M$.

□

Navazující studium

Mějte na paměti, že na pojmech množin, relací a funkcí jsou vystavěny prakticky všechny skutečné datové struktury používané v dnešní informatice, což můžete vidět na relačních databázích (viz také Lekce 6) a na často se vyskytujících induktivních definicích (Oddíl 6.5). S relacemi ekvivalence i jimi implicitně definovanými rozklady množin se lze setkat tam, kde nějaké objekty "rozdělujeme do přihrádek" podle nějakých sdílených znaků nebo jiných kritérií. Principy takových rozkladů vám zajisté byly intuitivně známy ještě dříve, než jste vůbec slyšeli o matematickém pojmu ekvivalence, v této lekci jsme si je jen uvedli na formálních základech. Toto formální pojetí relací ekvivalence a rozkladů budete potřebovat v různých navazujících předmětech, například u teorie automatů.

5 Uspořádané množiny, Uzávěry

Úvod

V této lekci dále pokračujeme probíráním binárních relací na množinách jako nástrojů vyjadřujících vztahy mezi objekty. Zaměřujeme se nyní především na relace „srovnávající" objekty podle jejich vlastností. Takto vágně opsané relace mívají jasné společné znaky, které se objevují ve zde uvedené formální definici relace uspořádání. Následně se také zabýváme způsobem, jak libovolnou relaci „obohatit" o nějakou vlastnost. Tento úkol vede na rozšiřování naší relace (tj. přidávání dvojic) do vzniku jejího takzvaného uzávěru.

Cíle

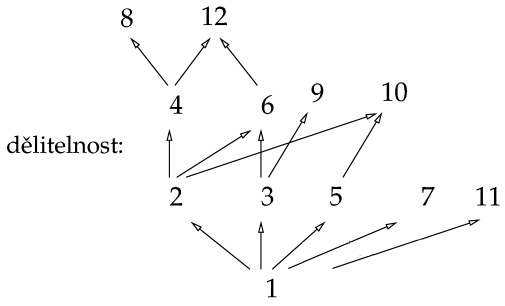
Definujeme relace uspořádání a mnohé další pojmy k nim se vztahující, například Hasseovy diagramy. Studující by měli porozumět matematické problematice uspořádaných množin a naučit se je správně používat. Závěrem si ukážeme operátory uzávěrů relací.

5.1 Uspořádání a uspořádané množiny

Podle Definice 4.4 je relace $R \subseteq M \times M$ *(částečné) uspořádání* právě když R je reflexivní, antisymetrická a tranzitivní. Tyto *tři vlastnosti* tedy musí být splněny a ověřeny k důkazu toho, že daná relace R je uspořádáním.

`Komentář:` Neformálně řečeno: uspořádání je taková relace $R \subseteq M \times M$, kde $(x, y) \in R$ právě když x je v nějakém smyslu „*menší nebo rovno*" než y. Mohou ovšem existovat taková $x, y \in M$, kde neplatí $(x, y) \in R$ ani $(y, x) \in R$. (Pak říkáme, že x a y jsou *nesrovnatelné*.)

Jak *názorně* zobrazit (částečné) uspořádání? Příklad zjednodušeného zakreslení (jsou vynechány šipky vyplývající z reflexivity a tranzitivity, viz Oddíl 5.3) je zde:



Všimněte si, že je někdy zvykem „větší" prvky kreslit nad ty „menší".

`Značení`. Pro větší přehlednost se relace uspořádání často značí symboly jako $\sqsubseteq$ či $\preceq$ místo prostého R. I my tak často budeme psát třeba $x \preceq y$ místo $(x, y) \in R$.

`Poznámka`: Zajisté jste se již neformálně setkali s „neostrým" uspořádáním čísel $\leq$ a „ostrým" uspořádáním $<$. Všimněte si dobře, že námi definované uspořádání je vždy „neostré". Pokud byste naopak chtěli definovat „*ostré*" uspořádání, mělo by vlastnosti *ireflexivní*, antisymetrické a tranzitivní. (Příliš se však tato varianta nepoužívá.) Nadále budeme pracovat pouze s neostrým uspořádáním, ale smyčky vyplývající z reflexivity budeme pro větší přehlednost v obrázcích vynechávat.

Uspořádaná množina

Matematicky důležitějším pojmem než samotná relace uspořádání je uspořádaná množina, neboli soubor prvků s pevně zvoleným uspořádáním na nich.

Definice 5.1. ***Uspořádaná množina*** je dvojice $(M, \preceq)$,
kde M je množina a $\preceq$ je (částečné) uspořádání na M.

`Definice`: Uspořádání $\preceq$ na M je *lineární* (nebo také *úplné*), pokud každé dva prvky M jsou v $\preceq$ srovnatelné.

Příklady uspořádaných množin

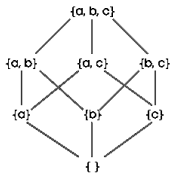
Příklad 5.2. *Nechť M je množina všech studentů 1. ročníku FI. Uvažme postupně relace uspořádání $R \subseteq M \times M$ definované následovně (jedná se vždy o uspořádání?):*

* $(x, y) \in R$ právě když x má alespoň takovou výšku jako y;
* $(x, y) \in R$ právě když y má alespoň takovou výšku jako x;
* $(x, y) \in R$ právě když x a y mají stejné rodné číslo.

Ano, i v posledním bodě se jedná o uspořádání! (Dobře si promyslete, proč. Které dvojice jsou vůbec porovnatelné?) □

Příklad 5.3. *Další ukázky uspořádaných množin následují zde:*

* $(\mathbb{N}, \leq)$ je lineárně uspořádaná množina, kde $\leq$ má „obvyklý“ význam.
* $(\mathbb{N}, |\,)$, kde $a|b$ je relace dělitelnosti „a dělí b“ na přirozených číslech, je uspořádaná množina. Toto uspořádání není lineární.
* Buď M množina. Pak $(2^M, \subseteq)$ je uspořádaná množina (říkáme *inkluzí*). Které dvojice jsou v uspořádání inkluzí *nesrovnatelné*?



□

`Komentář:` Zamyslete se také, jak vypadá relace dělitelnosti na celých (tj. i záporných) číslech. Proč se už nejedná o uspořádání? Blíže viz konec Oddílu 5.2.

Užitečnou dovedností je umět popsat uspořádání „větší množiny“ pomocí „malých“ složek. Neformálně lze toto uvést následujícími příklady:

Příklad 5.4. *Uspořádání „*po složkách*“.*

Nechť $(A, \leq_A)$ a $(B, \leq_B)$ jsou uspořádané množiny. Definujme binární relaci $\sqsubseteq$ na $A \times B$ předpisem

$$(a, b) \sqsubseteq (a', b') \quad \text{právě když} \quad a \leq_A a' \text{ a } b \leq_B b'.$$

Pak $(A \times B, \sqsubseteq)$ je uspořádaná množina. Toto uspořádání se nazývá „po složkách“. □

Příklad 5.5. Lexikografické *uspořádání.*

Nechť $(A, \leq_A)$ a $(B, \leq_B)$ jsou uspořádané množiny. Definujme binární relaci $\preceq$ na $A \times B$ předpisem

$$(a, b) \preceq (a', b') \quad \text{právě když} \quad \text{buď } a \leq_A a' \text{ a } a \neq a', \text{ nebo } a = a' \text{ a } b \leq_B b'.$$

Pak $(A \times B, \preceq)$ je uspořádaná množina. Navíc pokud $\leq_A$ i $\leq_B$ jsou lineární, je i $\preceq$ lineární. Toto uspořádání se nazývá lexikografické. □

`Fakt`: Jsou-li $(A_1, \leq_1), \cdots, (A_n, \leq_n)$ uspořádané množiny, kde $n \geq 2$, pak množinu $A_1 \times \cdots \times A_n$ lze uspořádat třeba *po složkách* nebo *lexikograficky*.

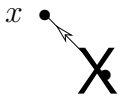
Všimněte si, že lexikograficky se například řadí slova ve slovníku...

5.2 Další pojmy uspořádaných množin

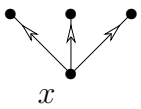
K tématu uspořádaných množin se vztahuje množství drobných pojmů, které potkáte v různých oblastech matematiky i informatiky.

Definice 5.6. Nechť $(M, \preceq)$ je uspořádaná množina.

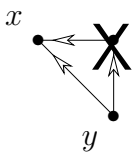
- $x \in M$ je *minimální* právě když pro každé $y \in M$ platí, že jestliže $y \preceq x$, pak $x \preceq y$. (Tj. x je minimální právě když neexistuje žádný prvek ostře menší než x.)



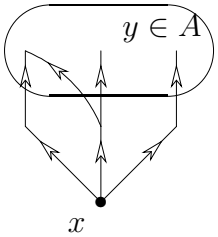
- $x \in M$ je *maximální* právě když pro každé $y \in M$ platí, že jestliže $x \preceq y$, pak $y \preceq x$. (Tj. x je maximální právě když neexistuje žádný prvek ostře větší než x.)
- $x \in M$ je *nejmenší* právě když pro každé $y \in M$ platí, že $x \preceq y$.



- $x \in M$ je *největší* právě když pro každé $y \in M$ platí, že $y \preceq x$.
- $x \in M$ *pokrývá* $y \in M$ právě když $x \neq y$, $y \preceq x$ a neexistuje žádné $z \in M$ takové, že $x \neq z \neq y$ a $y \preceq z \preceq x$.

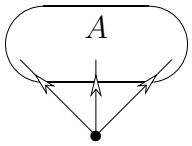


- $x \in M$ je *dolní závora* (mez) množiny $A \subseteq M$ právě když $x \preceq y$ pro každé $y \in A$.

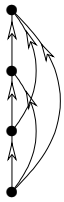


- $x \in M$ je *horní závora* (mez) množiny $A \subseteq M$ právě když $y \preceq x$ pro každé $y \in A$.

- $x \in M$ je *infimum* množiny $A \subseteq M$ právě když x je největší dolní závora množiny A.



- $x \in M$ je *supremum* množiny $A \subseteq M$, právě když x je nejmenší horní závora množiny A.
- $A \subseteq M$ je *řetězec* v uspořádání $\preceq$ právě když $(A, \preceq)$ je *lineárně* uspořádaná množina.



`Komentář:` Tuto dlouhou definici se sluší poněkud neformálně okomentovat. Za prvé, s pojmy nejmenšího a největšího prvku jste se už intuitivně setkali mnohokrát, ale (matematicky slabší) pojmy minimálního a maximálního působí někdy problémy. Zapamatujte si proto dobře, že minimálních prvků může mít množina několik, jsou to prostě všechny ty "vespod", ale nejmenší prvek existuje nejvýše jeden a je to pouze ten *unikátní* minimální prvek množiny. Stejně pro maximální...

Další poznámka se vztahuje k infimu a supremu množiny. Jak jsme napsali (a asi totéž znáte z matematické analýzy), množina nemusí mít nejmenší ani největší prvek, ale v mnoha případech je lze "nahradit" po řadě infimem a supremem, které hrají v jistých ohledech podobnou roli. Avšak ani supremum a infimum nemusí vždy existovat.

Pozor! Některé uvedené definice mají dosti „netriviální chování" na *nekonečných* množinách. Proto je budeme obvykle uvažovat jen nad konečnými množinami...

Příklad 5.7. *Proč má každá uspořádaná množina nejvýše jeden největší prvek?*

Tvrzení dokážeme sporem: Nechť m i n jsou největší prvky uspořádané množiny $(M, \preceq)$. Pak podle Definice 5.6 platí $n \preceq m$ i $m \preceq n$ zároveň. Ovšem jelikož uspořádání musí být antisymetrické, pak platí $m = n$ a největší prvek je jen jeden. □

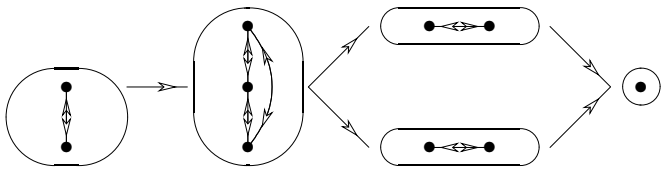
Relace předuspořádání

Mimo uspořádání chápaných striktně antisymetricky se v praxi často setkáme s „skorouspořádáními", ve kterých některé očividně různé prvky stojí na stejné úrovni naší preference. Jak třeba uspořádáte různé počítače podle výkonu nebo studenty podle známek A–E? To matematicky podchytí následující pojem.

`Definice`: Relace $R \subseteq M \times M$ je *předuspořádání* (také *kvaziuspořádání*, nebo *polouspořádání*) právě když R je *reflexivní* a *tranzitivní*.

`Komentář:` Rozdíl mezi uspořádáním a předuspořádáním je (neformálně řečeno!) v tom, že u předuspořádání srovnáváme prvky podle kritéria, které není pro daný prvek jedinečné. V předuspořádání takto mohou vznikat „*balíky*" (třídy) se stejnou hodnotou kritéria, což

schematicky ilustrujeme na následujícím obrázku.



Důležitým poznatkem je, že předuspořádání se „až tak moc“ neliší od dřívějšího uspořádání – přirozeně totiž odvodíme následující:

Věta 5.8. *Je-li $\sqsubseteq$ předuspořádání na M, můžeme definovat relaci $\sim$ na M předpisem*

$$x \sim y \quad \textit{právě když} \quad x \sqsubseteq y \text{ a } y \sqsubseteq x.$$

Pak $\sim$ je ekvivalence na M, která se nazývá jádro předuspořádání $\sqsubseteq$.

Na rozkladu $M/\sim$ pak lze zavést relaci $\preceq$ definovanou takto

$$[x] \preceq [y] \quad \textit{právě když } x \sqsubseteq y.$$

Pak $(M/\sim, \preceq)$ je uspořádaná množina indukovaná $\sqsubseteq$.

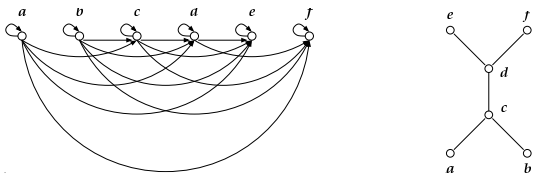
`Komentář:` Pro ukázku si vezměme relaci dělitelnosti na $\mathbb{Z}$. Pak třeba $-2 \sim 2$. Jádrem zde jsou dvojice čísel stejné absolutní hodnoty.

`Důkaz` (náznak): Tranzitivita a reflexivita relace $\sim$ vyplývá z tranzitivity a reflexivity relace $\sqsubseteq$. Symetrie $\sim$ pak je přímým důsledkem její definice. Tudíž $\sim$ skutečně je relací ekvivalence a $M/\sim$ je platný rozklad.

Tranzitivita a reflexivita relace $\preceq$ se opět dědí z relace $\sqsubseteq$. Její antisymetrie vyplývá následující úvahou: Pokud $[x] \preceq [y]$ a $[y] \preceq [x]$, pak podle naší definice $x \sqsubseteq y$ a $y \sqsubseteq x$, neboli $x \sim y$ a $[x] = [y]$ podle definice tříd rozkladu. Pozor, nejdůležitější částí této větve důkazu je však ještě zdůvodnění, že naše podaná definice vztahu $[x] \preceq [y]$ je korektní, což znamená, že její platnost nezávisí na konkrétní volbě reprezentantů x z $[x]$ a y z $[y]$. □

5.3 Hasseovské diagramy

Motivací zavedení tzv. Hasseovských diagramů uspořádaných množin jsou přehlednější „obrázky“ než u grafů relací. Například si srovnejte následující ukázky:

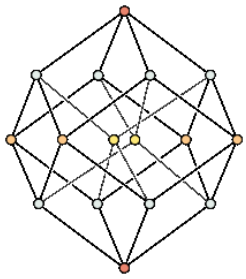


`Definice`: *Hasseovský diagram* konečné uspořádané množiny $(M, \preceq)$ je jeho (jednoznačné) grafické znázornění získané takto:

∗ Do první „horizontální vrstvy“ zakreslíme body odpovídající mininálním prvkům $(M, \preceq)$. (Tj. které *nepokrývají* nic. Pojem pokrývání prvku najdete v Definici 5.6.)

∗ Máme-li již zakreslenu vrstvu i, pak do vrstvy $i+1$ (která je „nad“ vrstvou i) zakreslíme všechny nezakreslené prvky, které *pokrývají pouze* prvky vrstev $\leq i$. Pokud prvek x vrstvy $i+1$ pokrývá prvek y vrstvy $\leq i$, spojíme x a y neorientovanou hranou (tj. „čárou“).

Příklad 5.9. *Relaci inkluze na čtyřprvkové množině* $\{a, b, c, d\}$ *zakreslíme Hasseovským diagramem takto:*



□

`Komentář:` Jak vidíme, v Hasseovském diagramu „*vynecháváme*“ ty hrany relace $\preceq$, které vyplývají z *reflexivity či tranzitivity*. To celý obrázek výrazně zpřehlední, a přitom nedochází ke ztrátě informace. Lze vynechat i šipky na hranách, neboť dle definice všechny míří „vzhůru“. Také pojem „*vrstvy*“ v definici je jen velmi neformální, důležité je, že větší (pokrývající) prvky jsou *nad menšími* (pokrývanými).

5.4 Uzávěry relací

Posledním bodem našeho výkladu je postup, jak danou relaci můžeme „obohatit“ o zvolenou vlastnost (například proto, že naše data o relaci jsou neúplná a vlastnost je tak poškozena).

`Definice`: Buď V (nějaká) vlastnost binárních relací. Řekneme, že V je *uzavíratelná*, pokud splňuje následující podmínky:

∗ Pro každou množinu M a každou relaci $R \subseteq M \times M$ existuje alespoň jedna relace $S \subseteq M \times M$, která má vlastnost V a pro kterou platí $R \subseteq S$.

∗ Nechť I je množina a nechť $R_i \subseteq M \times M$ je relace mající vlastnost V pro každé $i \in I$. Pak relace $\bigcap_{i \in I} R_i$ má vlastnost V.

`Fakt`: Libovolná kombinace vlastností *reflexivita, symetrie, tranzitivita* je uzavíratelná vlastnost. Antisymetrie není uzavíratelná vlastnost.

Věta 5.10. *Nechť V je uzavíratelná vlastnost binárních relací. Buď M množina a R libovolná binární relace na M. Pak pro množinu všech relací $S \supseteq R$ na M majících vlastnost V existuje infimum R_V (vzhledem k množinové inkluzi), které samo má vlastnost V.*

`Definice`: Tuto „nejmenší“ relaci R_V s vlastností V nazýváme *V-uzávěr* relace R.

Tvrzení 5.11. Nechť R je binární relace na M. Pak platí následující poznatky.

∗ *Reflexivní uzávěr* R je přesně relace $R \cup \{(x, x) \mid x \in M\}$.

∗ *Symetrický uzávěr* R je přesně relace $\overset{\leftrightarrow}{R} = \{(x, y) \mid (x, y) \in R \text{ nebo } (y, x) \in R\}$.

∗ *Tranzitivní uzávěr* R je přesně relace $R^+ = \bigcup_{i=1}^{\infty} \mathcal{T}^i(R)$, kde $\mathcal{T}$ je funkce, která pro každou binární relaci S vrátí relaci

$$\mathcal{T}(S) = S \cup \{(x,z) \mid \text{ existuje } y \text{ takové, že } (x,y),(y,z) \in S\}$$

a $\mathcal{T}^i = \underbrace{\mathcal{T} \circ \cdots \circ \mathcal{T}}_{i}$ je i-krát iterovaná aplikace funkce $\mathcal{T}$.

∗ Reflexivní a tranzitivní uzávěr R je přesně relace $R^* = Q^+$, kde Q je reflexivní uzávěr R.

∗ Reflexivní, symetrický a tranzitivní uzávěr R (tj. nejmenší ekvivalence obsahující R) je přesně relace $(\overset{\leftrightarrow}{Q})^+$, kde Q je reflexivní uzávěr R.

∗ Na pořadí aplikování uzávěrů vlastností záleží!

`Komentář:` Význam reflexivních a symetrických uzávěrů je z předchozího docela zřejmý. Význam tranzitivního uzávěru R^+ je následovný: Do R^+ přidáme všechny ty dvojice (x,z) takové, že v R se lze „dostat po šipkách“ z x do z. Nakreslete si to na papír pro nějakou jednoduchou relaci, abyste význam tranzitivního uzávěru lépe pochopili.

A jak bylo dříve řečeno, antisymetrický uzávěr relace prostě nemá smysl. Například buď $R \subseteq \mathbb{N} \times \mathbb{N}$ definovaná takto: $R = \{(i, i+1) \mid i \in \mathbb{N}\}$. Pak R^* je běžné lineární uspořádání $\leq$ přirozených čísel.

Příklad 5.12. *Proč při výpočtu tranzitivního uzávěru relace na konečné množině podle vzorce* $R^+ = \bigcup_{i=1}^{\infty} \mathcal{T}^i(R)$ *vždy stačí uvažovat konečně mnoho členů tohoto sjednocení?*

Pro odpověď si uvědomme zásadní fakt — pokud $\mathcal{T}^{i+1} = \mathcal{T}^i$, tak už platí $\mathcal{T}^{i+k} = \mathcal{T}^i$ pro všechna přirozená k. Neboli je potřeba sjednotit jen tolik prvních členů, dokud se ony „zvětšují“, což může nastat jen konečně krát nad konečnou množinou. Mimo jiné tak vidíme, že uvedený popis tranzitivního uzávěru je *konstruktivní*. □

Rozšiřující studium

I v této lekci se pojednává o látce, kterou určitě čtenáři na inuitivní úrovni znají (vždy umění si věci „uspořádat“ je jednou ze základních lidských dovedností). Přesto správné matematické uchopení podstaty uspořádání vyžaduje se prokousat několika nesnadnými formálními definicemi. Všimněte si, že hlavní těžkosti pojmu uspořádané množiny se vztahují k částečným, tedy ne-lineárním, uspořádáním, kdy běžnému lidskému uvažování není zcela intuitivně jasné nakládání s nesrovnatelnými dvojicemi.

Jmenovitě dále upozorňujeme na pojmy uzávěrů relace; co se týče praktického určení symetrického nebo tranzitivního uzávěru relace, odkazujeme na budoucí Algoritmy 8.5 a 8.6.

6 Skládání relací a funkcí

Úvod

Vraťme se nyní k látce Lekce 3. Z jejího pokročilého obsahu jsme doposud velmi detailně probírali relace a jejich jednotlivé vlastnosti. Nyní se podívejme, jak lze relace mezi sebou „skládat“, což je například základní technika práce s relačními databázemi. Je však i jiné místo, kde jste se zajisté se skládáním relací setkali – jedná se o skládání funkcí. Jak například spočítáte na kalkulačce výsledek složitějšího vzorce?

Mimo to se ještě zobecněně vrátíme k problematice „postupných“ (induktivních a rekurentních) definic a vztahů. Jedná se vlastně o matematické analogie rekurzivních programů a jejich správné formální pochopení oceníme mimo jiné i při programování samotném.

Cíle

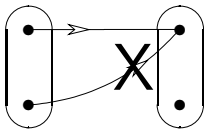
V této lekci definujeme základní vlastnosti funkcí a především popíšeme a podrobně rozebereme skládání relací a návazně skládání funkcí jako relací (jako model nám poslouží permutace). Na závěr se stručně podíváme na problematiku induktivních definic funkcí, coby na rozšíření rekurentních vztahů z Oddílu 3.4.

6.1 Vlastnosti funkcí

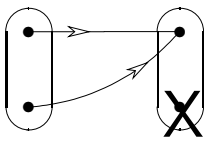
Funkce je podle Definice 4.2 speciálním případem (obvykle binární) relace, majícím pro každou hodnotu levé strany jedinou (funkční) hodnotu pravé strany. Více o funkcích řekne následující definice.

Definice 6.1. ***Funkce*** (případně parciální funkce) $f : A \to B$ je

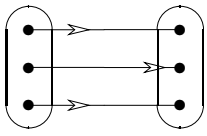
* *injektivní* (nebo také *prostá*) právě když pro každé $x, y \in A$, $x \neq y$ platí $f(x) \neq f(y)$;



* *surjektivní* (nebo také „*na*“) právě když pro každé $y \in B$ existuje $x \in A$ takové, že $f(x) = y$;



* *bijektivní* (vzáj. *jednoznačná*) právě když je injektivní a *současně* surjektivní.



Funkce nás coby speciální případ relací budou provázet i ve zbytku lekce.

`Komentář:` Následují jiné ukázky vlastností funkcí.

* Funkce *plus* : $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ je surjektivní, ale není prostá.

* Funkce $g : \mathbb{Z} \to \mathbb{N}$ daná předpisem

$$g(x) = \begin{cases} -2x - 1 & \text{jestliže } x < 0, \\ 2x & \text{jinak} \end{cases}$$

je bijektivní.

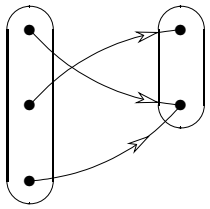
* Funkce $\emptyset : \emptyset \to \emptyset$ je bijektivní.
* Funkce $\emptyset : \emptyset \to \{a, b\}$ je injektivní, ale není surjektivní.
* Dokázali byste nalézt bijektivní funkci $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$?

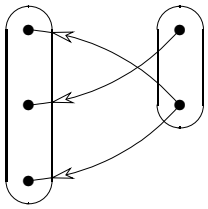
6.2 Inverzní relace a skládání relací

K použitelné práci s relacemi v aplikacích se dostaneme, pokud budeme umět relace správně „skládat“ a „převracet“. K tomu nás přivedou zase následující definice.

`Definice`: Nechť $R \subseteq A \times B$ je *binární* relace mezi A a B. *Inverzní relace* k relaci R se značí R^{-1} a je definována takto:

$$R^{-1} = \{(b, a) \mid (a, b) \in R\}$$

R: 

R^{-1}: 

R^{-1} je tedy relace *mezi* B a A!

`Komentář:` Příklady inverzí pro relace–funkce.

* Inverzí *bijektivní* funkce $f(x) = x + 1$ na $\mathbb{Z}$ je funkce $f^{-1}(x) = x - 1$.
* Inverzí *prosté* funkce $f(x) = e^x$ na $\mathbb{R}$ je *parciální* funkce $f^{-1}(x) = \ln x$.
* Funkce $g(x) = x \bmod 3$ *není prostá* na $\mathbb{N}$, a proto její inverzí je *„jen“* relace $g^{-1} = \{(a, b) \mid a = b \bmod 3\}$. Konkr. $g^{-1} = \{(0,0), (0,3), (0,6), \ldots, (1,1), (1,4), \ldots, (2,2), (2,5), \ldots\}$.

Tvrzení 6.2. *Mějme funkci $f : A \to B$. Pak její inverzní relace f^{-1} je*

a) parciální funkce právě když f je prostá,

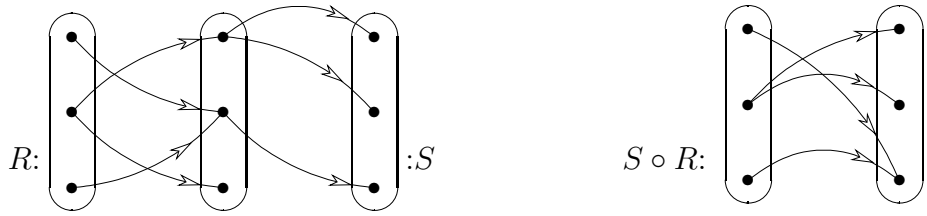
b) funkce právě když f je bijektivní.

`Důkaz` vyplývá přímo z definic funkce a inverze relace. □

Následuje klíčová definice, pro jejíž bližší ilustraci odkazujeme také na Příklad 6.4.

Definice 6.3. ***Složení (kompozice)*** relací R a S.
Nechť $R \subseteq A \times B$ a $S \subseteq B \times C$ jsou binární relace. *Složení* relací R a S (v tomto pořadí!) je relace $S \circ R \subseteq A \times C$ definovaná takto:

$$S \circ R = \{(a, c) \mid \text{existuje } b \in B \text{ takové, že } (a, b) \in R, (b, c) \in S\}$$



Složení relací čteme „*R složeno s S*“ nebo (pozor na pořadí!) „*S po R*“.

`Komentář:` Několik matematických příkladů skládání relací následuje zde.

* Je-li
  - $A = \{a, b\}, \quad B = \{1, 2\}, \quad C = \{X, Y\},$
  - $R = \{(a, 1), (b, 1), (b, 2)\}, \quad S = \{(1, X)\},$

  pak složením vznikne relace
  - $S \circ R = \{(a, X), (b, X)\}.$
* Složením funkcí $h(x) = x^2$ a $f(x) = x + 1$ na $\mathbb{R}$ vznikne funkce

$$(f \circ h)(x) = f(h(x)) \; = \; x^2 + 1.$$

* Složením těchže funkcí „*naopak*“ ale vznikne funkce $(h \circ f)(x) = h(f(x)) \; = \; (x + 1)^2$.

`Poznámka`: Nepříjemné je, že v některých oblastech matematiky (například v algebře při skládání zobrazení) se setkáme s *právě opačným* zápisem skládání, kdy se místo $S \circ R$ píše $R \cdot S$ nebo jen RS. Proto je si vždy dobré slovně ujasnit, které pořadí skládaných relací máme na mysli. My zde zásadně budeme používat pořadí $S \circ R$.

6.3 Skládání relací „v praxi“

Podívejme se nyní, jak se skládání relací přirozeně objevuje v práci s relačními databázemi. (Dá se zjednodušeně říci, že právě v operátoru skládání tabulkových relací tkví hlavní smysl relačních databází...)

Příklad 6.4. *Skládání v relační databázi studentů, jejich předmětů a fakult.*

Mějme dvě binární relace – jednu R přiřazující studentům MU kódy jejich zapsaných předmětů, druhou S přiřazující kódy předmětů jejich mateřským fakultám. Malý výsek z těchto relací může v tabulkové reprezentaci vypadat třeba následovně.

R :

student (učo)	předmět (kód)
121334	MA010
133935	M4135
133935	IA102
155878	M1050
155878	IB000

S :

předmět (kód)	fakulta MU
MA010	FI
IB000	FI
IA102	FI
M1050	PřF
M4135	PřF

Jak z těchto „tabulkových“ relací zjistíme, kteří studenti mají zapsané předměty na kterých fakultách (třeba na FI)?

Jedná se jednoduše o *složení relací* $S \circ R$. V našem příkladě tabulkové reprezentace vyjde výsledek:

$S \circ R$:

student (učo)	fakulta MU
121334	FI
133935	FI
133935	PřF
155878	FI
155878	PřF

□

Zobecněné skládání relací

V praktických použitích relačních tabulek povětšinou nevystačíme jen s binárními relacemi, takže je přirozené se ptát, jestli lze podobně skládat i více-ární relace. Odpověď je snadná – lze to a ani nepotřebujeme novou definici, vystačíme s tou, kterou už máme výše uvedenou.

`Definice`: (***skládání relací vyšší arity***):
Mějme relace $T \subseteq K_1 \times K_2 \times \cdots \times K_k$ a $U \subseteq L_1 \times L_2 \times \cdots \times L_\ell$, přičemž pro nějaké $m < \min(k, \ell)$ platí $L_1 = K_{k-m+1}, L_2 = K_{k-m+2}, \ldots, L_m = K_k$. Pak relaci T lze *složit* s relací U na zvolených m složkách $L_1, \ldots, L_m$ („překrytí“) s *použitím Definice 6.3* takto:

* Položme $A = K_1 \times \cdots \times K_{k-m}$, $B = L_1 \times \cdots \times L_m$ a $C = L_{m+1} \times \cdots \times L_\ell$.
* Příslušné relace pak jsou $R = \{(\vec{a}, \vec{b}) \in A \times B \mid (a_1, \ldots a_{k-m}, b_1, \ldots b_m) \in T\}$ a $S = \{(\vec{b}, \vec{c}) \in B \times C \mid (b_1, \ldots b_m, c_{m+1}, \ldots c_\ell) \in U\}$.
* Nakonec přirozeně položme $U \circ_m T \simeq S \circ R$, takže vyjde

$$U \circ_m T = \left\{(\vec{a}, \vec{c}) \mid \text{ ex. } \vec{b} \in B, \text{ že } (a_1, \ldots a_{k-m}, b_1, \ldots b_m) \in T \text{ a } (b_1, \ldots b_m, c_{m+1}, \ldots c_\ell) \in U\right\}.$$

Schematicky pro snažší orientaci ve složkách našich relací:

$$\begin{array}{rlll} T \subseteq & K_1 \times \cdots \times K_{k-m} \times & K_{k-m+1} \times \cdots \times K_k & \\ U \subseteq & & L_1 \quad \times \cdots \times L_m & \times L_{m+1} \times \cdots \times L_\ell \\ U \circ_m T \subseteq & \underbrace{K_1 \times \cdots \times K_{k-m}}_{A} \times & \underbrace{\qquad\qquad}_{B} & \times \underbrace{L_{m+1} \times \cdots \times L_\ell}_{C} \end{array}$$

Opět je nejjednodušší si koncept skládání vícečetných relací ilustrovat příkladem.

Příklad 6.5. *Skládání v relační databázi pasažérů a letů u leteckých společností.*

Podívejme se na příklad hypotetické rezervace letů pro cestující, relace T. Jak známo (tzv. codeshare), letecké společnosti si mezi sebou „dělí“ místa v letadlech, takže různé lety (podle kódů) jsou ve skutečnosti realizovány stejným letadlem jedné ze společností. To zase ukazuje relace U.

T :

pasažér	datum	let
Petr	5.11.	OK535
Pavel	6.11.	OK535
Jan	5.11.	AF2378
Josef	5.11.	DL5457
Alena	6.11.	AF2378

U :

datum	let	letadlo
5.11.	OK535	ČSA
5.11.	AF2378	ČSA
5.11.	DL5457	ČSA
6.11.	OK535	AirFrance
6.11.	AF2378	AirFrance

Ptáme-li se nyní, setkají se Petr a Josef na palubě stejného letadla? Případně, čí letadlo to bude? Odpovědi nám dá složení relací $U \circ_2 T$, jak je posáno výše.

$U \circ_2 T$:

pasažér	letadlo
Petr	ČSA
Josef	ČSA
Pavel	AirFrance
...	...

Zkuste se zamyslet, lze tyto dvě relace skládat ještě jinak? Co by pak bylo významem? □

6.4 Skládání funkcí, permutace

Soustřeďme se nyní na další oblast, kde běžně a přirozeně používáme skládání relací, aniž si to uvědomujeme.

`Fakt`: Mějme zobrazení (funkce) $f : A \to B$ a $g : B \to C$. Pak jejich *složením* coby relací v tomto pořadí vznikne zobrazení $(g \circ f) : A \to C$ definované

$$(g \circ f)(x) = g(f(x)) \,.$$

`Komentář:`

* Jak například na běžné kalkulačce vypočteme hodnotu funkce $\sin^2 x$? Složíme (v tomto pořadí) „elementární“ funkce $f(x) = \sin x$ a $g(x) = x^2$.
* Jak bychom na „elementární“ funkce rozložili aritmetický výraz $2\log(x^2 + 1)$? Ve správném pořadí složíme funkce $f_1(x) = x^2$, $f_2(x) = x + 1$, $f_3(x) = \log x$ a $f_4(x) = 2x$.
* A jak bychom obdobně vyjádřili složením funkcí aritmetický výraz $\sin x + \cos x$? Opět je odpověď přímočará, vezmeme „elementární“ funkce $g_1(x) = \sin x$ a $g_2(x) = \cos x$, a pak je „složíme“ další funkcí $h(x, y) = x + y$. Vidíme však, že takto pojaté „skládání“ už nezapadá hladce do našeho zjednodušeného formalismu skládání relací.

  Pro nedostatek prostoru si skládání funkcí s více parametry nedefinujeme, ale sami vidíte, že obdobné skládání se v programátorské praxi vyskytuje doslova „na každém rohu“ a ani se nad tím nepozastavujeme.

Skládání permutací

Po zbytek tohoto oddílu se zaměříme na permutace coby speciální případ (bijektivních) zobrazení.

`Definice`: Nechť *permutace* π množiny $\{1, 2, \ldots, n\}$ je určena seřazením jejích prvků $(p_1, \ldots, p_n)$. Pak π je zároveň *bijektivním zobrazením* $\{1, \ldots, n\} \to \{1, \ldots, n\}$ definovaným předpisem $\pi(i) = p_i$. Tudíž lze permutace *skládat jako relace* podle Definice 6.3.

`Poznámka`: Všechny permutace množiny $\{1, 2, \ldots, n\}$ spolu s operací skládání tvoří grupu, zvanou symetrická grupa S_n. Permutační grupy (podgrupy symetrické grupy) jsou velice důležité v algebře, neboť každá grupa je vlastně isomorfní některé permutační grupě.

`Komentář:` Příkladem permutace vyskytujícím se v programátorské praxi je třeba zobrazení $i \mapsto (i+1) \mod n$ ("inkrement"). Často se třeba lze setkat (aniž si to mnohdy uvědomujeme) s permutacemi při indexaci prvků polí.

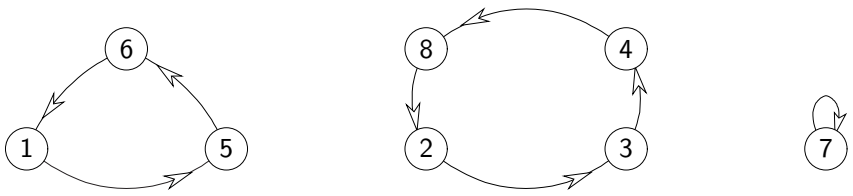
V kontextu pohledu na funkce a jejich skládání coby relací si zavedeme jiný, názornější, způsob zápisu permutací – pomocí jejich cyklů.

`Definice`: Nechť π je permutace na množině A. *Cyklem v* π rozumíme posloupnost $\langle a_1, a_2, \ldots, a_k \rangle$ různých prvků A takovou, že $\pi(a_i) = a_{i+1}$ pro $i = 1, 2, \ldots, k-1$ a $\pi(a_k) = a_1$.

Jak název napovídá, v zápise cyklu $\langle a_1, a_2, \ldots, a_k \rangle$ není důležité, kterým prvkem začneme, ale jen dodržení cyklického pořadí. Cyklus v permutaci může mít i jen jeden prvek (zobrazený na sebe).

`Komentář:` Nakreslete si (vámi zvolenou) permutaci π obrázkem, ve kterém vedete šipku vždy od prvku i k prvku $\pi(i)$. Pak uvidíte, že cykly dle naší definice jsou právě cykly tvořené šipkami ve vašem obrázku. S tímto grafickým zobrazením pro vás nebude problém pochopit následující látku.

Například permutaci $(5, 3, 4, 8, 6, 1, 7, 2)$ si lze obrázkem nakreslit takto:



Reprezentace permutací jejich cykly

Věta 6.6. *Každou permutaci π na konečné množině A lze zapsat jako složení cyklů na disjunktních podmnožinách (rozkladu) A.*

`Důkaz`: Vezmeme libovolný prvek $a_1 \in A$ a iterujeme zobrazení $a_2 = \pi(a_1)$, $a_3 = \pi(a_2)$, atd., až se dostaneme „zpět" k $a_{k+1} = \pi(a_k) = a_1$. Proč tento proces skončí? Protože A je konečná a tudíž ke zopakování některého prvku a_{k+1} musí dojít. Nadto je π prostá, a proto nemůže nastat $\pi(a_k) = a_j$ pro $j > 1$. Takto získáme první cyklus $\langle a_1, \ldots, a_k \rangle$.

Induktivně pokračujeme s hledáním dalších cyklů ve zbylé množině $A \setminus \{a_1, \ldots, a_k\}$, dokud nezůstane prázdná. □

`Značení` permutace jejími cykly: Nechť se permutace π podle Věty 6.6 skládá z cyklů $\langle a_1, \ldots, a_k \rangle$, $\langle b_1, \ldots, b_l \rangle$ až třeba $\langle z_1, \ldots, z_m \rangle$. Pak zapíšeme

$$\pi = (\, \langle a_1, \ldots, a_k \rangle \, \langle b_1, \ldots, b_l \rangle \ldots \langle z_1, \ldots, z_m \rangle \,) \,.$$

`Komentář:` Primitivní pseudonáhodné generátory v počítačích iterují z náhodného počátku permutaci danou vztahem $i \mapsto (i+p) \mod q$. Je pochopitelné, že tato permutace nesmí obsahovat krátké cykly, lépe řečeno, měla by se skládat z jediného (dlouhého) cyklu. (Pro úplnost, jedná se o permutaci množiny $\{0, 1, \ldots, q-1\}$).

Příklad 6.7. *Ukázka skládání permutací daných svými cykly.*

Vezměme 7-prvkovou permutaci $\pi = (3, 4, 5, 6, 7, 1, 2)$. Ta se skládá z jediného cyklu $\langle 1, 3, 5, 7, 2, 4, 6 \rangle$. Jiná permutace $\sigma = (5, 3, 4, 2, 6, 1, 7)$ se rozkládá na tři cykly $\langle 1, 5, 6 \rangle$,

$\langle 2,3,4\rangle$ a $\langle 7\rangle$. Nyní určíme složení $\sigma \circ \pi$ těchto dvou permutací (už přímo v zápisu cykly):

$$(\langle 1,5,6\rangle\langle 2,3,4\rangle\langle 7\rangle) \circ (\langle 1,3,5,7,2,4,6\rangle) = (\langle 1,4\rangle\langle 2\rangle\langle 3,6,5,7\rangle)$$

(Nezapomínejme, že první se ve složení aplikuje pravá permutace!)
Postup skládání jsme použili následovný: 1 se zobrazí v permutaci vpravo na 3 a pak vlevo na 4. Následně 4 se zobrazí na 6 a pak na 1. Tím „uzavřeme“ první cyklus $\langle 1,4\rangle$. Dále se 2 zobrazí na 4 a pak hned zpět na 2, tj. má samostatný cyklus. Zbylý cyklus $\langle 3,6,5,7\rangle$ určíme analogicky. □

6.5 Induktivní definice množin a funkcí

Vzpomeňme si na definici posloupnosti *rekurentním vztahem* z Oddílu 3.4. Přímým zobecněním dřívějších rekurentních definic je následující koncept.

Definice 6.8. ***Induktivní definice*** množiny.
Jedná se obecně o popis (nějaké) množiny M v následujícím tvaru:

- Je dáno několik pevných (*bázických*) prvků $a_1, a_2, \ldots, a_k \in M$.
- Je dán soubor *induktivních pravidel* typu

  Jsou-li (libovolné prvky) $x_1, \ldots, x_\ell \in M$, pak také $y \in M$.

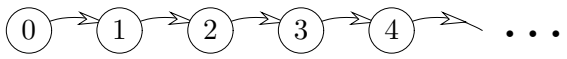
  V tomto případě je y typicky funkcí $y = f_i(x_1, \ldots, x_\ell)$.

Pak naše *induktivně definovaná množina* M je určena jako nejmenší (inkluzí) množina vyhovující těmto pravidlům.

`Poznámka`: Vidíte podobnost této definice s uzávěrem relace? (Věta 5.10.)

`Komentář:`

- Pro nejbližší příklad induktivní definice se obrátíme na množinu všech přirozených čísel, která je formálně zavedena následovně.
  - $0 \in \mathbb{N}$
  - Je-li $i \in \mathbb{N}$, pak také $i + 1 \in \mathbb{N}$.

- Pro každé $y \in \mathbb{N}$ můžeme definovat jinou množinu $M_y \subseteq \mathbb{N}$ induktivně takto:
  - $y \in M_y$
  - Jestliže $x \in M_y$ a $x + 1$ je liché, pak $x + 2 \in M_y$.

  Pak například $M_3 = \{3\}$, nebo $M_4 = \{4 + 2i \mid i \in \mathbb{N}\}$.

Jednoznačnost induktivních definic

`Definice`: Řekneme, že daná induktivní definice množiny M je *jednoznačná*, právě když každý prvek M lze odvodit z bázických prvků pomocí induktivních pravidel právě *jedním způsobem*.

`Komentář:` Definujme například množinu $M \subseteq \mathbb{N}$ induktivně takto:
- $2, 3 \in M$
- Jestliže $x, y \in M$ a $x \leq y$, pak také $x^2 + y^2$ a $x \cdot y$ jsou prvky M.

Proč tato induktivní definice není jednoznačná? Například číslo $8 \in M$ lze odvodit způsobem $8 = 2 \cdot (2 \cdot 2)$, ale zároveň zcela jinak $8 = 2^2 + 2^2$.

V čem tedy spočívá důležitost jednoznačných induktivních definic množin? Je to především v dalším možném využití induktivní definice množiny jako „základny“ pro odvozené vyšší definice, viz následující.

Definice 6.9. ***Induktivní definice funkce*** z induktivní množiny.
Nechť množina M je dána jednoznačnou induktivní definicí. Pak říkáme, že funkce $\mathcal{F} : M \to X$ je definována *induktivně* (vzhledem k induktivní definici M), pokud je řečeno:

- Pro každý z bázických prvků $a_1, a_2, \ldots, a_k \in M$ je určeno $\mathcal{F}(a_i) = c_i$, kde c_i je konstanta.
- Pro každé induktivní pravidlo typu

  “Jsou-li (libovolné prvky) $x_1, \ldots, x_\ell \in M$, pak také $f(x_1, \ldots, x_\ell) \in M$”

  je definováno

  $$\mathcal{F}\big(f(x_1, \ldots, x_\ell)\big) \text{ na základě hodnot } \mathcal{F}(x_1), \ldots, \mathcal{F}(x_\ell).$$

`Komentář:` Pro příklad se podívejme třeba do manuálových stránek unixového příkazu `test EXPRESSION`:

```
EXPRESSION is true or false and sets exit status.  It is one of:
 ! EXPRESSION                  EXPRESSION is false
 EXPRESSION1 -a EXPRESSION2    both EXPRESSION1 and EXPRESSION2 are true
 EXPRESSION1 -o EXPRESSION2    either EXPRESSION1 or EXPRESSION2 is true
 [-n] STRING                   the length of STRING is nonzero
 STRING1 = STRING2             the strings are equal
   ......
```

Vidíte, jak tato ukázka koresponduje s Definicí 6.9 ?

Induktivní definice se „strukturální“ indukcí

Závěrem ještě doplňkově zařazujeme malou ukázku, kterak přirozeně zkombinovat induktivní definice s *pokročilou formou matematické indukce* v dokazování, s tzv. *strukturální indukcí*.

Příklad 6.10. *Jednoduché aritmetické výrazy a jejich význam.*

Nechť je dána abeceda $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \odot, \oplus, (,)\}$. Definujme množinu *jednoduchých výrazů SExp* $\subseteq \Sigma^*$ induktivně takto:

– Dekadický zápis každého přirozeného čísla $\mathbf{n}$ je prvek *SExp*.
– Jestliže $x, y \in$ *SExp*, pak také $(x) \odot (y)$ a $(x) \oplus (y)$ jsou prvky *SExp*. (Jak vidíme, díky nucenému závorkování je tato induktivní definice jednoznačná.)

Tímto jsme aritmetickým výrazům přiřadili jejich „formu“, tedy *syntaxi*.

Pro přiřazení „významu“, tj. *sémantiky* aritmetického výrazu, následně definujme funkci $\mathit{Val} : \mathit{SExp} \to \mathbb{N}$ induktivně takto:

– Bázické prvky: $\mathit{Val}(\mathbf{n}) = n$, kde $\mathbf{n}$ je dekadický zápis přirozeného čísla n.
– První induktivní pravidlo: $\mathit{Val}((x) \oplus (y)) = \mathit{Val}(x) + \mathit{Val}(y)$.
– Druhé induktivní pravidlo: $\mathit{Val}((x) \odot (y)) = \mathit{Val}(x) \cdot \mathit{Val}(y)$.

Co je tedy správným významem uvedených aritmetických výrazů? (Příklad 6.11) □

Příklad 6.11. *Důkaz správnosti přiřazeného „významu“ $Val : SExp \to \mathbb{N}$.*

> **Věta**. *Pro každý výraz $s \in SExp$ je hodnota $Val(s)$ z Příkladu 6.10 číselně rovna výsledku vyhodnocení výrazu s podle běžných zvyklostí aritmetiky.*

Jelikož pojednáváme o induktivně definované funkci *Val*, je přirozené pro důkaz jejích vlastností aplikovat *matematickou indukci*. Na rozdíl od dříve probíraných příkladů zde nevidíme žádný celočíselný „parametr n“, a proto si jej budeme muset nejprve definovat. Naši indukci tedy povedeme podle „délky ℓ odvození výrazu s“ definované jako *počet aplikací induktivních pravidel* potřebných k odvození $s \in SExp$.

`Důkaz`: V *bázi indukce* ověříme vyhodnocení bázických prvků, kteréžto jsou zde dekadické zápisy přirozených čísel. Platí $Val(\mathbf{n}) = n$, což skutečně odpovídá zvyklostem aritmetiky.

V *indukčním kroku* se podíváme na vyhodnocení $Val((x) \oplus (y)) = Val(x) + Val(y)$. Podle běžných zvyklostí aritmetiky by hodnota $Val((x) \oplus (y))$ měla být rovna *součtu* vyhodnocení *výrazu* x, což je podle indukčního předpokladu rovno $Val(x)$ (x má zřejmě kratší délku odvození), a vyhodnocení *výrazu* y, což je podle indukčního předpokladu rovno $Val(y)$. Takže skutečně $Val((x) \oplus (y)) = Val(x) + Val(y)$.

Druhé pravidlo $Val((x) \odot (y))$ se dořeší analogicky. □

Rozšiřující studium

S formalizací pojmu funkce a jejími vlastnostmi se setkáváte především v matematice, avšak například na bijektivní funkce narazíte při ukládání dat při volbě klíče apod. Pro informatiku jsou spíše důležitější relace a jejich skládání, *na nichž je založena mimo jiné práce s relačními databázemi. Poslední částí lekce je problematika induktivních definic, které ač ve formálním podání mohou nejprve vypadat nepochopitelně, jsou ve skutečnosti zcela přirozenou popisnou metodou v mnoha aplikačních sférách informatiky a jejich alespoň intuitivní chápání pro vás bude v dalším studiu nezbytné. Schválně, zkuste se podívat zrovna do vaší oblíbené oblasti informatiky, kde všude* induktivní definice *(třebaže nepřímo) najdete.*

7 Jemný úvod do Logiky

Úvod

Základem přesného matematického vyjadřování (viz Lekce 1) je správné používání matematické logiky a logických úsudků. Logika jako filozofická disciplína se intenzivně vyvíjí už od dob antiky, avšak ke skutečnému rozmachu formální logiky coby součásti matematiky došlo až začátkem 20. století. (S přispěním třeba Russelova paradoxu a převratných Gödelových prací.)

Dnes se základní logický kalkulus používá nejen v čisté matematice, ale „stojí" na něm veškeré logické obvody a počítače. Proto se studenti informatiky s logikou setkávají záhy při svém studiu a mnohokrát se k tématu také vracejí.

Cíle

Následující lekce přináší (skutečně velmi jemný a tak trochu i povrchní) úvod do moderní matematické logiky, která je solidním základem matematiky a nakonec i celé informatiky. Zavedeme si základy výrokové logiky a výrokového počtu a velmi stručně si uvedeme problematiku predikátové logiky a kvantifikace.

7.1 Výroky v přirozené podobě

Prvním úkolem našeho výukového materiálu je vytvořit pevný „most" mezi přirozenou lidskou mluvou a formálním jazykem matematiky. To napomůže pochopení významu matematických výroků a práci s nimi.

`Definice`: V přirozené mluvě za *výrok* považujeme (každé) tvrzení, o kterém má smysl prohlásit, že je buď pravdivé nebo nepravdivé.

`Komentář:` Ukážeme si několik příkladů – které z nich jsou výroky?

- Dnes už v Brně pršelo.
- Předmět FI: IB000 se vyučuje v prvním ročníku.
- Platí $2 + 3 = 6$.
- To je bez problémů. *(Co?)*
- Platí $x > 3$.
- Pro každé celé číslo x platí, že $x > 3$.

Všimněte si, že pravdivost výroku by mělo být možné rozhodnout bez skrytých souvislostí (kontextu), a proto čtvrtý a pátý příklad za výroky nepovažujeme. Poslední příklad již zase výrokem je, neboť obor hodnot x je jednoznačně vymezen tzv. kvantifikací.

Z více jednoduchých výroků vytváříme výroky složitější pomocí tzv. *logických spojek*.

`Komentář:` Následuje několik dalších příkladů.

- Kateřina přijela ve 12:00 a šli jsme spolu do kina.
- Množina $\{a, b\}$ má více než jeden prvek a není nekonečná.
- Jestliže má Karel přes 90 kg váhy, nejedu s ním výtahem.
- Jestliže má tato kráva 10 nohou, mají všechny domy modrou střechu.

Zastavme se na chvíli nad posledním výrokem. Co nám říká? Je pravdivý? Skutečně mají všechny domy modrou střechu a před námi stojí kráva s 10 nohama? Tuto problematiku jsme již neformálně nakousli v Lekci 1 a přesnou odpověď poskytne Definice 7.3. (Ano, výrok je pravdivý.)

Schopnost porozumět podobným větám je součást lidského způsobu uvažování a z tohoto hlediska nemá přímou souvislost s matematikou (je to „*přirozená logika*“). *Formální (matematická) logika* pak v podobném duchu definuje jazyk matematiky a přitom odstraňuje nejednoznačnosti přirozeného jazyka.

7.2 Formální výroková logika

Pro dokončení přechodu z přirozené mluvy do formálního matematického jazyka je nutno dát druhému rigorózní náplň. Tato náplň formálního jazyka se skládá ze *syntaxe* („*jak to zapíšeme*“) a *sémantiky* („*co to znamená*“).

Definice 7.1. ***Syntaxe výrokové logiky.***
Buď $\mathcal{AT} = \{A, B, C, \dots\}$ spočetně nekonečná množina *výrokových proměnných* (tzv. atomů). Množina *výrokových formulí* Φ je *definována induktivně* následujícími pravidly:

(1) $\mathcal{AT} \subseteq \Phi$.

(2) Jestliže $\varphi, \psi \in \Phi$, pak také $\neg(\varphi) \in \Phi$ a $(\varphi) \Rightarrow (\psi) \in \Phi$.

(3) Každý prvek Φ vznikne konečně mnoha aplikacemi pravidel (1) a (2).

`Značení`: Symbol $\neg$ je zván *negací* a $\Rightarrow$ je nazýván *implikací*.

`Komentář:` Příklady několika správně utvořených formulí:

$$A, \quad (A) \Rightarrow (B), \quad ((A) \Rightarrow (\neg(B))) \Rightarrow ((\neg(B)) \Rightarrow (C))$$

A také příklady několika ne zcela správně utvořených formulí:

$$A \Rightarrow B, \quad A \Rightarrow B \Rightarrow C, \quad \neg A \Rightarrow B$$

Z těchto nesprávných ukázek je „nejhorší“ ta prostřední, neboť daný zápis neumožňuje ani odhadnout, která z implikací se má vyhodnotit jako první. Abychom si ujasnili, která zjednodušení formálního zápisu formule jsou ještě přijatelná beze ztráty smyslu, přistoupíme k následující dohodě.

Konvence 7.2. Pro zvýšení čitelnosti budeme závorky vynechávat, pokud to nepovede k nejednoznačnostem za předpokladu, že negace $\neg$ má „vyšší prioritu“ než $\Rightarrow$. (Touto úmluvou se nemění množina Φ; mění se jen *způsob reprezentace* jejích prvků.)

Dále si zavedeme, že

* $\varphi \vee \psi$ (*disjunkce*) je jiný zápis formule $\neg\varphi \Rightarrow \psi$,
* $\varphi \wedge \psi$ (*konjunkce*) je jiný zápis formule $\neg(\neg\varphi \vee \neg\psi)$,
* $\varphi \Leftrightarrow \psi$ (*ekvivalence*) je jiný zápis formule $(\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$.

`Komentář:` Například formule $\big(\neg((A) \Rightarrow (B))\big) \Rightarrow \big((\neg(B)) \Rightarrow (C)\big)$ se dá s naší konvencí zapsat jako $(A \Rightarrow B) \vee B \vee C$.

Definice 7.3. ***Sémantika výrokové logiky.***
Valuace (ohodnocení) je funkce $\nu : \mathcal{AT} \to \{true, false\}$. Pro každou valuaci ν definujeme funkci $\mathcal{S}_\nu : \Phi \to \{true, false\}$ (*vyhodnocení*) induktivně takto:

* $\mathcal{S}_\nu(A) = \nu(A)$ pro každé $A \in \mathcal{AT}$.
* $\mathcal{S}_\nu(\neg\varphi) = \begin{cases} true & \text{jestliže } \mathcal{S}_\nu(\varphi) = false; \\ false & \text{jinak.} \end{cases}$

* $\mathcal{S}_\nu(\varphi \Rightarrow \psi) = \begin{cases} \mathit{false} & \text{jestliže } \mathcal{S}_\nu(\varphi) = \mathit{true} \text{ a } \mathcal{S}_\nu(\psi) = \mathit{false}; \\ \mathit{true} & \text{jinak.} \end{cases}$

`Poznámka`: Tento předpis podává nejen *definici* funkce $\mathcal{S}_\nu$, ale také návod na to, jak ji pro daný argument *vypočítat*.

Tvrzení 7.4. *Důsledkem Definice 7.3 je následovné:*

* $\mathcal{S}_\nu(\varphi \vee \psi) = \mathit{true}$ právě když $\mathcal{S}_\nu(\varphi) = \mathit{true}$ nebo $\mathcal{S}_\nu(\psi) = \mathit{true}$.
* $\mathcal{S}_\nu(\varphi \wedge \psi) = \mathit{true}$ právě když $\mathcal{S}_\nu(\varphi) = \mathit{true}$ a současně $\mathcal{S}_\nu(\psi) = \mathit{true}$.
* $\mathcal{S}_\nu(\varphi \Leftrightarrow \psi) = \mathit{true}$ právě když platí jedna z následujících podmínek
  - $\mathcal{S}_\nu(\varphi) = \mathit{true}$ a současně $\mathcal{S}_\nu(\psi) = \mathit{true}$,
  - $\mathcal{S}_\nu(\varphi) = \mathit{false}$ a současně $\mathcal{S}_\nu(\psi) = \mathit{false}$.

Pravdivostní tabulky

V praxi často vyhodnocení $\mathcal{S}_\nu$ logické výrokové formule zapisujeme do tzv. *pravdivostní tabulky*. Tato tabulka typicky má sloupce pro jednotlivé proměnné, případné „meziformule“ (pomůcka pro snažší vyplnění) a výslednou formuli. Řádků je 2^p (počet valuací), kde p je počet použitých proměnných. Místo $\mathit{true}, \mathit{false}$ píšeme $1, 0$.

Pro naše účely postačí uvést pravdivostní tabulku instruktážním příkladem. Čtenáři nechť si vyplní podle Definice 7.3 vlastní pravdivostní tabulky dalších výrokových formulí, viz také příslušné cvičící odpovědníky IS MU.

Příklad 7.5. *Jaká je pravdivostní tabulka pro formuli* $(A \Rightarrow B) \vee B \vee C$*?*

A	B	C	$A \Rightarrow B$	$(A \Rightarrow B) \vee B \vee C$
0	0	0	1	1
0	1	0	1	1
1	0	0	0	0
1	1	0	1	1
0	0	1	1	1
0	1	1	1	1
1	0	1	0	1
1	1	1	1	1

□

`Definice`: Formule $\varphi \in \Phi$ je *splnitelná*, pokud pro <u>některou</u> valuaci ν platí, že $\mathcal{S}_\nu(\varphi) = \mathit{true}$. Formule $\varphi \in \Phi$ je *vždy pravdivá*, neboli výroková *tautologie*, psáno $\models \varphi$, pokud pro <u>každou</u> valuaci ν platí, že $\mathcal{S}_\nu(\varphi) = \mathit{true}$.

Řekneme, že dvě formule $\varphi, \psi \in \Phi$ jsou *ekvivalentní*, právě když $\models \varphi \Leftrightarrow \psi$.

Tvrzení 7.6. *Několik užitečných tautologií:*

* $\models A \vee \neg A$
* $\models \neg\neg A \Leftrightarrow A$
* $\models (A \wedge (A \Rightarrow B)) \Rightarrow B$
* $\models (\neg B \Rightarrow \neg A) \Rightarrow (A \Rightarrow B)$

* $\models (\neg A \Rightarrow (B \wedge \neg B)) \Rightarrow A$

`Komentář:` Kde jsme se s užitím takových tautologií už setkali? A jak poznáme tautologii v pravdivostní tabulce?

7.3 Jak správně znegovat formuli?

Přesný význam formulí se zanořenými negacemi je někdy obtížné zjistit (podobně jako v běžné řeči).

„Není pravda, že nemohu neříct, že není pravda, že tě nemám nerad."

Výrokové formule se proto obvykle prezentují v tzv. normálním tvaru, kde se negace vyskytují pouze u výrokových proměnných, formálně:

`Definice`: Formule $\varphi \in \Phi$ je v *normálním tvaru*, pokud se v ní operátor negace aplikuje pouze na výrokové proměnné.

`Komentář:` Například, pokud přijmeme pravidlo „dvojí negace" ($\neg\neg A \Leftrightarrow A$), tak výše napsanou větu si převedeme na lépe srozumitelný tvar:

„Nemusím říct, že tě mám nerad."

Tvrzení 7.7. *Každou výrokou formuli lze převést do normálního tvaru, pokud k $\Rightarrow$ povolíme i užívání odvozených spojek $\wedge$ a $\vee$.*

`Komentář:` Pro ilustraci k $\neg(A \Rightarrow B)$ je ekvivalentní $A \wedge \neg B$, k $\neg\big(C \wedge (\neg A \Rightarrow B)\big)$ je ekvivalentní $\neg C \vee (\neg A \wedge \neg B)$ a k $\neg\big((A \Rightarrow B) \Rightarrow C\big)$ je ekvivalentní $(A \Rightarrow B) \wedge \neg C)$.

`Poznámka`: Při dalším studiu logiky či logického programování se setkáte s různými jinými „normálními formami" výrokových formulí, například s konjunktivní či disjunktivní normální formou. S naší definicí mají všechny důležitý styčný bod – použití negací jen u proměnných.

Normální tvar (formální postup)

„*Znegováním formule φ*" se obvykle myslí převod její negace $\neg\varphi$ do normálního tvaru. Jak jsme již demonstrovali na příkladě, normální tvar formule je „výrazně čitelnější" než formule používající zanořené negace. Proto následující formální postup induktivně definuje právě způsob převodu libovolné výrokové formule do našeho normálního tvaru.

Metoda 7.8. ***Převod formule φ do normálního tvaru $\mathcal{F}(\varphi)$.***
Definujeme funktory $\mathcal{F}$ a $\mathcal{G}$ pro náš převod induktivními předpisy

$$
\begin{array}{lcllcl}
\mathcal{F}(A) & = & A & \mathcal{G}(A) & = & \neg A \\
\mathcal{F}(\neg\varphi) & = & \mathcal{G}(\varphi) & \mathcal{G}(\neg\varphi) & = & \mathcal{F}(\varphi) \\
\mathcal{F}(\varphi \Rightarrow \psi) & = & \mathcal{F}(\varphi) \Rightarrow \mathcal{F}(\psi) & \mathcal{G}(\varphi \Rightarrow \psi) & = & \mathcal{F}(\varphi) \wedge \mathcal{G}(\psi) \\
\mathcal{F}(\varphi \wedge \psi) & = & \mathcal{F}(\varphi) \wedge \mathcal{F}(\psi) & \mathcal{G}(\varphi \wedge \psi) & = & \mathcal{G}(\varphi) \vee \mathcal{G}(\psi) \\
\mathcal{F}(\varphi \vee \psi) & = & \mathcal{F}(\varphi) \vee \mathcal{F}(\psi) & \mathcal{G}(\varphi \vee \psi) & = & \mathcal{G}(\varphi) \wedge \mathcal{G}(\psi) \\
\mathcal{F}(\varphi \Leftrightarrow \psi) & = & \mathcal{F}(\varphi) \Leftrightarrow \mathcal{F}(\psi) & \mathcal{G}(\varphi \Leftrightarrow \psi) & = & (\mathcal{F}(\varphi) \wedge \mathcal{G}(\psi)) \vee (\mathcal{G}(\varphi) \wedge \mathcal{F}(\psi))
\end{array}
$$

`Komentář:` Uvažme formuli $\neg(A \Rightarrow \neg(B \vee \neg(C \Rightarrow \neg A)))$. Užitím postupu 7.8 získáme:

$$\begin{array}{lcll} \mathcal{F}(\neg(A \Rightarrow \neg(B \vee \neg(C \Rightarrow \neg A)))) & = & \mathcal{G}(A \Rightarrow \neg(B \vee \neg(C \Rightarrow \neg A))) & = \\ \mathcal{F}(A) \wedge \mathcal{G}(\neg(B \vee \neg(C \Rightarrow \neg A))) & = & A \wedge \mathcal{F}(B \vee \neg(C \Rightarrow \neg A)) & = \\ A \wedge (\mathcal{F}(B) \vee \mathcal{F}(\neg(C \Rightarrow \neg A))) & = & A \wedge (B \vee \mathcal{G}(C \Rightarrow \neg A)) & = \\ A \wedge (B \vee (\mathcal{F}(C) \wedge \mathcal{G}(\neg A))) & = & A \wedge (B \vee (C \wedge \mathcal{F}(A))) & = \\ A \wedge (B \vee (C \wedge A)) & & & \end{array}$$

Formuli $A \wedge (B \vee (C \wedge A))$ lze dále zjednodušit na (ekvivalentní) formuli $A \wedge (B \vee C)$. To ale je již z našeho pohledu matematicky neformální (heuristický) postup.

Uvedené formální předpisy takto vyjadřují „*intuitivní postup negace*“ v matematicky přesném tvaru. Důkaz tohoto tvrzení je zároveň velmi hezkou ukázkou použití strukturální matematické indukce.

Věta 7.9. *Pro libovolnou výrokovou formuli φ platí, že*

a) $\mathcal{F}(\varphi)$ je ekvivalentní formule k φ v normálním tvaru

b) a $\mathcal{G}(\varphi)$ je formule v normálním tvaru ekvivalentní negaci $\neg\varphi$.

`Důkaz` povedeme tzv. „*indukcí ke struktuře formule*“, tedy indukci povedeme podle „délky“ ℓ – počtu aplikací induktivních pravidel (Definice 7.1) při sestavování formule φ.

- Báze indukce ($\ell = 0$): Pro všechny atomy, tj. výrokové proměnné, zřejmě platí, že $\mathcal{F}(A) = A$ je ekvivalentní A a $\mathcal{G}(A) = \neg A$ je ekvivalentní $\neg A$.
- V indukčním kroku předpokládejme, že a) i b) platí pro všechny formule φ délky nejvýše ℓ. Vezmeme si formuli ψ délky $\ell + 1$, která je utvořená jedním z následujících způsobů:
  - $\psi \equiv \neg\varphi$ ($\equiv$ je „definiční rovnítko“ pro formule). Podle výše uvedeného induktivního předpisu je $\mathcal{F}(\psi) = \mathcal{F}(\neg\varphi) = \mathcal{G}(\varphi)$. Podle indukčního předpokladu pak je $\mathcal{G}(\varphi)$ formule v normálním tvaru ekvivalentní $\neg\varphi = \psi$. Obdobně pro funktor $\mathcal{G}$ vyjádříme $\mathcal{G}(\psi) = \mathcal{G}(\neg\varphi) = \mathcal{F}(\varphi)$. Podle indukčního předpokladu pak je $\mathcal{F}(\varphi)$ formule v normálním tvaru ekvivalentní φ a to je dále ekvivalentní $\neg\neg\varphi = \neg\psi$ podle Tvrzení 7.6.
  - $\psi \equiv (\varphi_1 \Rightarrow \varphi_2)$. Podle výše uvedeného induktivního předpisu je $\mathcal{F}(\psi) = \mathcal{F}(\varphi_1 \Rightarrow \varphi_2) = \mathcal{F}(\varphi_1) \Rightarrow \mathcal{F}(\varphi_2)$. Podle indukčního předpokladu jsou $\mathcal{F}(\varphi_1)$ i $\mathcal{F}(\varphi_2)$ formule v normálním tvaru ekvivalentní φ_1 a φ_2. Potom i $\mathcal{F}(\varphi_1) \Rightarrow \mathcal{F}(\varphi_2)$ je v normálním tvaru dle definice a podle sémantiky $\Rightarrow$ je ta ekvivalentní formuli $(\varphi_1 \Rightarrow \varphi_2) = \psi$.
    Obdobně rozepíšeme $\mathcal{G}(\psi) = \mathcal{G}(\varphi_1 \Rightarrow \varphi_2) = \mathcal{F}(\varphi_1) \wedge \mathcal{G}(\varphi_2)$. Jelikož $\wedge$ je pro nás jen zkratka, výraz dále rozepíšeme $\mathcal{G}(\psi) = \neg(\mathcal{F}(\varphi_1) \Rightarrow \neg\mathcal{G}(\varphi_2))$. Podle indukčního předpokladu (a dvojí negace) jsou $\mathcal{F}(\varphi_1)$ a $\neg\mathcal{G}(\varphi_2)$ po řadě ekvivalentní formulím φ_1 a φ_2. Tudíž nakonec odvodíme, že $\mathcal{G}(\psi)$ je ekvivalentní negaci formule $\varphi_1 \Rightarrow \varphi_2$, což jsme zde měli dokázat.
  - $\psi \equiv (\varphi_1 \vee \varphi_2)$. Zde si musíme opět uvědomit, že spojka $\vee$ je pro nás jen zkratka, a přepsat $\psi \equiv (\neg\varphi_1 \Rightarrow \varphi_2)$. Potom podle předchozích dokázaných případů víme, že $\mathcal{F}(\psi) = \mathcal{F}(\neg\varphi_1 \Rightarrow \varphi_2) = \mathcal{F}(\neg\varphi_1) \Rightarrow \mathcal{F}(\varphi_2)$ je ekvivalentní formuli $(\neg\varphi_1 \Rightarrow \varphi_2) = \psi$, což bylo třeba dokázat. Stejně tak $\mathcal{G}(\psi) = \mathcal{G}(\neg\varphi_1 \Rightarrow \varphi_2) = \mathcal{F}(\neg\varphi_1) \wedge \mathcal{G}(\varphi_2)$ je podle předchozích případů důkazu ekvivalentní $(\neg\varphi_1 \wedge \neg\varphi_2) = \neg\psi$.
  - $\psi \equiv (\varphi_1 \wedge \varphi_2)$ a $\psi \equiv (\varphi_1 \Leftrightarrow \varphi_2)$ už dokončíme analogicky. □

7.4 Predikátová logika, kvantifikace

Výše popsaná výroková logika je velmi omezená faktem, že každý výrok musí být („absolutně“) vyhodnocen jako pravda nebo nepravda. Co když však chceme zpracovat tvrzení typu „den D v Brně pršelo“? Jeho pravdivostní hodnota přece závisí na tom, co dosadíme za den D, a tudíž jej nelze považovat za výrok výrokové logiky.

- *Predikátová logika* je *obecnější než logika výroková*; každá formule výrokové logiky je i formulí predikátové logiky, ale ne obráceně.
- Predikátová logika pracuje s *predikáty*. Predikáty jsou „*parametrizované výroky*“, které jsou buď pravdivé nebo nepravdivé pro každou konkrétní volbu parametrů. Výrokové proměnné lze chápat jako predikáty bez parametrů.

  `Komentář:` Pro neformální přiblížení si uvedeme několik ukázek predikátů:
  - $x > 3$ (parameterem je zde $x \in \mathbb{R}$),
  - R je ekvivalence na M (parametr $R \subseteq M \times M$),
  - čísla x a y jsou nesoudělná (parametry $x, y \in \mathbb{N}$),
  - obecně jsou predikáty psány $P(x, y)$, kde x, y jsou libovolné parametry.

Definice 7.10. ***Syntaxe i sémantika predikátové logiky.***
Z predikátů lze vytvářet *predikátové formule* pomocí už známých (viz Definice 7.1) výrokových spojek a následujících tzv. *kvantifikátorů*:

- $\forall x \,.\, \varphi$ „pro *každou* volbu parametru x platí formule φ“,
- $\exists x \,.\, \varphi$ „*existuje* alespoň jedna volba parametru x, pro kterou platí φ“.

`Fakt`: Je-li *každá* proměnná – parametr predikátu – v dané formuli kvantifikovaná (tj. formule je *uzavřená*), pak je celá formule buď pravdivá nebo nepravdivá.

Konvence 7.11. Pro lepší srozumitelnost zápisu formulí predikátové logiky se domluvíme na následujícím.

- „Neuzavřené“ proměnné vyskytující se v predikátech ve formuli φ někdy pro referenci vypisujeme jako „parametry“ samotné formule $\varphi(x, y, \dots)$.
- Poukud není z kontextu jasné, co lze za daný parametr dosazovat, užívá se notace $\forall x \in M \,.\, \varphi$ a $\exists x \in M \,.\, \varphi$. (Platí, jen pokud je kvantifikovaný parametr prvkem nějaké fixní množiny!).
- Tečka za symbolem kvantifikátoru se někdy vynechává (při vhodném uzávorkování formule), nebo se používá symbol „ : “.
- Místo $\forall x_1 \,.\, \forall x_2 \,.\, \cdots \forall x_n \,.\, \varphi$ se někdy krátce píše $\forall x_1, x_2, \cdots, x_n \ (\varphi)$.
  Podobně u existenčního kvantifikátoru $\exists x_1, x_2, \cdots, x_n \ (\varphi)$.

Příklad 7.12. *Ukažme si vyjádření následujících slovních výroků v predikátové logice:*

- Každé prvočíslo větší než 2 je liché;

  $$\forall n \in \mathbb{N} \,.\ (Pr(n) \wedge n{>}2) \ \Rightarrow \ Li(n).$$

- Každé číslo $n > 1$, které není prvočíslem, je dělitelné nějakým číslem y kde $n \neq y$ a $y > 1$;

$$\forall n \in \mathbb{N}\,.\ \big(n > 1 \wedge \neg Pr(n)\big) \Rightarrow\ \exists y\big(y|\, n \wedge n{\neq}y \wedge y{>}1\big).$$

- Jsou-li R a S ekvivalence na M, je také $R \cup S$ ekvivalence na M. Zde například můžeme mít dva pohledy na toto tvrzení – v jednom bereme množinu M za pevnou

  $$\forall R, S\ :\ (Eq_M(R) \wedge Eq_M(S))\ \Rightarrow\ Eq_M(R{\cup}S),$$

  kdežto ve druhém je i množina M parametrem

  $$\forall M\ \forall R, S\ :\ (Eq(M,R) \wedge Eq(M,S))\ \Rightarrow\ Eq(M, R{\cup}S).$$

□

Příklad 7.13. *V pytlíku máme různobarevné kuličky. Vytáhneme libovolné tři z nich. Zamyslete se, proč je pak matematicky pravdivé následující tvrzení:*

- *Je-li první kulička (z vytažených) větší než druhá a zároveň je druhá kulička větší než první, tak je třetí kulička červená.*

Řešení: Tento příklad je podobný Příkladu 1.4. Opět lze „selským rozumem“ namítat, že je to přece úplná blbost – nemůže být první kulička větší než druhá a zároveň naopak, také nemůže být pravda, že třetí vytažená kulička je vždy červená. Jenže je potřeba se na uvedené tvrzení podívat matematicky.

V prvé řadě, slova „libovolné tři“ (z kuliček) vyjadřují všeobecnou kvantifikaci ($\forall$). Neboli uvedené tvrzení má být pravdivé pro každou trojici kuliček. Tvrzení samo je vysloveno jako implikace, takže je pravdivé vždy s výjimkou případů, kdy jsou splněny předpoklady a nepravdivý závěr. Jenže předpoklady (první kulička větší než druhá a zároveň naopak) nejsou splněny pro žádnou trojici kuliček a tudíž je naše tvrzení univerzálně platné. □

Jak „negovat“ formule predikátové logiky?

Na závěr jen stručně rozšíříme výše uvedenou metodu formálního negování.

Metoda 7.14. ***Převod predikátové formule φ do normálního tvaru $\mathcal{F}(\varphi)$.*** *Dřívější Metodu 7.8 rozšíříme o následující indukční pravidla:*

$$\begin{array}{lcllcl}
\mathcal{F}(P(x_1,\ldots,x_n)) & = & P(x_1,\ldots,x_n) & \mathcal{G}(P(x_1,\ldots,x_n)) & = & \neg P(x_1,\ldots,x_n) \\
\mathcal{F}(\forall x\,.\,\varphi) & = & \forall x\,.\,\mathcal{F}(\varphi) & \mathcal{G}(\forall x\,.\,\varphi) & = & \exists x\,.\,\mathcal{G}(\varphi) \\
\mathcal{F}(\exists x\,.\,\varphi) & = & \exists x\,.\,\mathcal{F}(\varphi) & \mathcal{G}(\exists x\,.\,\varphi) & = & \forall x\,.\,\mathcal{G}(\varphi)
\end{array}$$

`Komentář:` Uvažme například negaci výše uvedené formule

$$\neg(\forall M\ \forall R, S\ :\ (Eq(M,R) \wedge Eq(M,S))\ \Rightarrow\ Eq(M, R{\cup}S))\,.$$

Pak

$$\begin{array}{l}
\mathcal{F}(\neg(\forall M\ \forall R, S\ :\ (Eq(M,R) \wedge Eq(M,S))\ \Rightarrow\ Eq(M, R{\cup}S))) \ = \\
\mathcal{G}(\forall M\ \forall R, S\ :\ (Eq(M,R) \wedge Eq(M,S))\ \Rightarrow\ Eq(M, R{\cup}S)) \ = \\
\exists M\ \exists R, S\ :\ \mathcal{G}((Eq(M,R) \wedge Eq(M,S))\ \Rightarrow\ Eq(M, R{\cup}S)) \ = \\
\exists M\ \exists R, S\ :\ (Eq(M,R)\ \wedge\ Eq(M,S)\ \wedge\ \neg Eq(M, R{\cup}S))\,.
\end{array}$$

Rozšiřující studium

Základem každého počítače jsou logické obvody, které jsou jen praktickou funkční implementací vhodných formulí výrokové logiky. Tudíž se dá směle říci, že bez výrokové logiky by nebylo ani počítačů, ani informatiky. Nadto se informatici setkávají po teoretické stránce s výrokovou i predikátovou logikou nejen v matematice, ale i v logickém programování a nakonec i při psaní obyčejných složených podmínek v příkazu „`if`*“...*

Na druhou stranu zájemce o studium matematických hlubin logiky a třeba slavných Gödelových poznatků můžeme později odkázat na předmět Matematická logika MA007.

8 Dokazování vlastností algoritmů

Úvod

Po předchozí převážně matematické látce se náš výklad obrací zase k informatice a navazuje na druhý pilíř celého předmětu z Lekce 1 – algoritmy a jejich formální zápis. Jak jste asi již (i v našem předmětu) poznali, umění programovat není zdaleka jen o tom naučit se syntaxi programovacího jazyka, ale především o schopnosti vytvářet a správně formálně zapisovat algoritmy. Přitom situace, kdy programátorem zapsaný kód ve skutečnosti počítá něco trochu jiného, než si programátor představuje, je snad nejčastější programátorskou chybou – o to zákeřnější, že ji žádný „chytrý" překladač nemůže odhalit.

Proto již na počátku (seriózního) studia informatiky je dobré klást důraz na správné chápání zápisu algoritmů i na důkazy jejich vlastností a správnosti. A to je téma, kterému se tedy budeme věnovat po převážný zbytek předmětu FI:IB000.

Cíle

Na podkladě formálního zápisu algoritmů z Oddílu 1.4 si ukážeme, jak využít předchozí nabyté matematické znalosti při dokazování vlastností a správnosti různých jednoduchých algoritmů. Nejčastějším nástrojem nám při tom bude matematická indukce.

Neformálně o „správnosti" programů

Jak se máme přesvědčit, že je daný program „správný"?

* Co třeba ladění programů? Jelikož počet možných vstupních hodnot je (v principu) neohraničený, *nelze otestovat* všechna možná vstupní data.
* Situace je zvláště komplikovaná v případě paralelních, randomizovaných, interaktivních a nekončících programů (operační systémy, systémy řízení provozu apod.). Takové systémy mají *nedeterministické chování* a opakované experimenty vedou k různým výsledkům. (Nelze je tudíž ani rozumně ladit, respektive ladění poskytne jen velmi nedostatečnou záruku správného chování za jiných okolností...)
* V některých případech je však třeba mít naprostou jistotu, že program funguje tak jak má, případně že splňuje základní bezpečnostní požadavky.
  - Pro „malé" algoritmy je možné podat přesný matematický důkaz správnosti.
  - Narůstající složitosti programových systémů a požadavky na jejich bezpečnost si pak vynucují vývoj jiných „spolehlivých" formálních *verifikačních metod*.

8.1 Jednoduché indukční dokazování

Pro svěžení znalostí se nejprve podívejte zpět na naše konvence formálního zápisu algoritmů do Oddílu 1.4. Náš výklad začneme s několika zcela jednoduchými algoritmy, jejichž jediný účel je v demonstraci využití matematické indukce pro dokazování vlastností.

Příklad 8.1. *Zjistěte, kolik znaků* `'x'` *v závislosti na celočíselné hodnotě n vstupního parametru* `n` *vypíše následující algoritmus.*

Algoritmus 8.2.

```
foreach  i ← 1,2,3,...,n-1,n  do
    foreach  j ← 1,2,3,...,i-1,i  do
```

```
        vytiskni  'x';
    done
done
```

Nejprve si uvědomíme, že druhý (vnořený) cyklus vždy vytiskne celkem i znaků `'x'`. Proto iterací prvního cyklu (nejspíše) dostaneme postupně $1 + 2 + \cdots + n$ znaků `'x'` na výstupu, což již víme (Příklad 2.7), že je celkem $\frac{1}{2}n(n+1)$. Budeme tedy dokazovat následující tvrzení:

Věta. Pro každé přirozené n Algoritmus 8.2 vypíše právě $\frac{1}{2}n(n+1)$ znaků `'x'`.

Důkaz: Postupujeme indukcí podle n. Báze pro $n = 0$ je zřejmá, neprovede se ani jedna iterace cyklu a tudíž bude vytištěno 0 znaků `'x'`, což je správně.

Nechť tedy tvrzení platí pro jakékoliv n_0 a položme $n = n_0 + 1$. Prvních n_0 iterací vnějšího cyklu podle indukčního předpokladu vypíše (ve vnitřním cyklu) celkem $\frac{1}{2}n_0(n_0+1)$ znaků `'x'`. Pak již následuje jen jedna poslední iterace vnějšího cyklu s `i` $\leftarrow$ `n=`n_0`+1` a v ní se vnitřní cyklus `j` $\leftarrow$ `1,2,...,i=n` iteruje celkem $n = n_0 + 1$ -krát. Celkem tedy bude vytištěn tento počet znaků `'x'`:

$$\frac{1}{2}n_0(n_0+1) + n_0 + 1 = \frac{1}{2}(n_0+1+1)(n_0+1) = \frac{1}{2}n(n+1)$$

Důkaz indukčního kroku je hotov. □

Příklad 8.3. *Zjistěte, kolik znaků* `'z'` *v závislosti na celočíselné hodnotě* n *vstupního parametru* `n` *vypíše následující algoritmus.*

Algoritmus 8.4.

```
st ← "z";
foreach  i ← 1,2,3,...,n-1,n  do
    vytiskni řetězec  st;
    st  ←  st . st ;    (zřetězení dvou kopií st za sebou)
done
```

Zkusíme-li si výpočet simulovat pro $n = 0, 1, 2, 3, 4 \ldots$, postupně dostaneme počty `'z'` jako $0, 1, 3, 7, 15 \ldots$. Na základě toho již není obtížné „uhodnout“, že počet `'z'` bude (asi) obecně určen vztahem $2^n - 1$. Toto je však třeba dokázat!

Komentář: Jak záhy zjistíme, matematická indukce na naše tvrzení přímo „nezabírá“, ale mnohem lépe se nám povede s následujícím přirozeným zesílením dokazovaného tvrzení:

Věta. Pro každé přirozené n Algoritmus 8.4 vypíše právě $2^n - 1$ znaků `'z'` a proměnná `st` bude na konci výpočtu obsahovat řetězec 2^n znaků `'z'`.

Důkaz: Postupujeme indukcí podle n. Báze pro $n = 0$ je zřejmá, neprovede se ani jedna iterace cyklu a tudíž bude vytištěno $0 = 2^0 - 1$ znaků `'z'`, což bylo třeba dokázat. Mimo to proměnná `st` iniciálně obsahuje $1 = 2^0$ znak `'z'`.

Nechť tedy tvrzení platí pro jakékoliv n_0 a položme $n = n_0 + 1$. Podle indukčního předpokladu po prvních n_0 iteracích bude vytištěno $2^{n_0} - 1$ znaků `'z'` a proměnná `st` bude obsahovat řetězec 2^{n_0} znaků `'z'`. V poslední iteraci cyklu (pro `i` $\leftarrow$ `n=`n_0`+1`) vytiskneme dalších 2^{n_0} znaků `'z'` (z proměnné `st`) a dále řetězec `st` „zdvojnásobíme“. Proto po n iteracích bude vytištěno celkem $2^{n_0} - 1 + 2^{n_0} = 2^{n_0+1} - 1 = 2^n - 1$ znaků `'z'` a v `st` bude uloženo $2 \cdot 2^{n_0} = 2^{n_0+1} = 2^n$ znaků `'z'`. □

8.2 Algoritmy pro relace

V dalších pokročilejších ukázkách volíme algoritmy pro relace, které představují velice vhodnou strukturou pro algoritmické zpracování. (Pilní studenti si zajisté dokážou navrhnout sami i další podobné algoritmy.)

Algoritmus 8.5. ***Symetrický uzávěr.***
Pro danou relaci R na n-prvkové množině $A = \{a_1, a_2, \ldots, a_n\}$ vytvoříme její symetrický uzávěr $\overset{\leftrightarrow}{R}$ takto:

$\overset{\leftrightarrow}{R} \leftarrow R$;
`foreach i ← 1,2,...,n-1,n do`
    `foreach j ← 1,2,...,n-1,n do`
        `if` $(a_i, a_j) \in R \wedge (a_j, a_i) \notin R$ `then` $\overset{\leftrightarrow}{R} \leftarrow \overset{\leftrightarrow}{R} \cup \{(a_j, a_i)\}$;
    `done`
`done`

`Důkaz`: Zde není důkaz vůbec obtížný. Relace $\overset{\leftrightarrow}{R} \supseteq R$ je zřejmě symetrická, neboť (vnitřní) tělo cyklu pro všechny dvojice $(a_i, a_j) \in R$ přidá i (a_j, a_i). Z druhé strany všechny dvojice „přidané“ v $\overset{\leftrightarrow}{R} \setminus R$ musí být obsaženy podle definice symetrické relace, takže $\overset{\leftrightarrow}{R}$ je skutečně symetrickým uzávěrem podle definice uzávěru relace. □

`Poznámka`: Všimněte si, že jsme tento algoritmus zapsali *abstraktně* – vůbec jsme nekonkretizovali datové typy a struktury pro zápis relací. (Náš algoritmus jako takový tyto konkretizace nepotřebuje.) Na jednu stranu je to výhodné, neboť algoritmus můžeme pak použít na libovolně implementovan relace. V neposlední řadě je takový abstraktní zápis i přehlednější a snadněji pochopitelný. Na druhou stranu však programátor sám teď musí zvolit vhodnou implementaci relace, například jako dvourozměrné pole `R`, kde `R[i,j]=0` právě když $(a_i, a_j) \notin R$. Následně je třeba zdůvodnit i *správnost zvolené implementace*(!).

Algoritmus 8.6. ***Tranzitivní uzávěr.***
Pro danou relaci R na n-prvkové množině $A = \{a_1, a_2, \ldots, a_n\}$ vytvoříme její tranzitivní uzávěr R^+ takto:

$R^+ \leftarrow R$;
`foreach k ← 1,2,...,n-1,n do`
    `foreach i ← 1,2,...,n-1,n do foreach j ← 1,2,...,n-1,n do`
        `if` $(a_i, a_k) \in R^+ \wedge (a_k, a_j) \in R^+$ `then`
            `if` $(a_i, a_j) \notin R^+$ `then` $R^+ \leftarrow R^+ \cup \{(a_i, a_j)\}$;
        `fi`
    `done`
`done`

Jak by se dala dokázat správnost popsaného algoritmu? Přímá aplikace indukce podle n nevypadá přínosně... (Zkuste si sami!) Nejkratší cesta k cíli vede použitím indukce (podle proměnné k vnějšího cyklu) na vhodně zesíleném tvrzení. Pro jeho formulaci si definujeme, že relace S na A je *k-částečně tranzitivní*, pokud pro libovolná i, j a pro $\ell \leq k$ platí, že z $(a_i, a_\ell), (a_\ell, a_j) \in S$ vyplývá $(a_i, a_j) \in S$. (Všimněte si, že pro $k = 0$ tato definice neříká nic a pro $k = n$ znamená běžnou tranzitivní relaci.)

> **Věta.** Po každých $k \geq 0$ iteracích vnějšího cyklu Algoritmu 8.6 aktuální hodnota relace R^+ udává k-částečně tranzitivní uzávěr relace R na A.

`Důkaz`: Báze indukce pro $k = 0$ jasně platí, neboť věta v tom případě nic neříká.

Předpokládejme nyní, že tvrzení platí pro nějaké $k_0 \geq 0$ a dokažme jej i pro $k = k_0+1$. Zřejmě stačí uvažovat případ $k_0 < n$. Každá dvojice (a_i, a_j) přidaná do R^+ uvnitř cyklů musí náležet do k-částečně tranzitivního uzávěru podle definice. Zbývá zdůvodnit, proč každá dvojice (a_i, a_j) náležející do k-částečně tranzitivního uzávěru, ale ne do k_0-částečně tranzitivního uzávěru, bude do R^+ v k-té iteraci přidána.

Není těžké ověřit, že (a_i, a_j) náleží do k-částečně tranzitivního uzávěru, právě když v relaci R nalezneme takovou cestu „po šipkách“ z a_i do a_j, která přechází pouze přes prvky a_ℓ kde $\ell \leq k$. V naší situaci vyplývá, že taková cesta musí použít i prvek a_k (jen jednou!), a proto (a_i, a_k) i (a_k, a_j) náleží do k_0-částečně tranzitivního uzávěru R. V k-té iteraci tudíž bude příslušná `if` podmínka splněná a (a_i, a_j) bude přidána do R^+. □

8.3 Dokazování konečnosti algoritmu

`Komentář:` Všimněte si, že jsme se zatím v důkazech vůbec nezamýšleli nad tím, zda náš algoritmus vůbec skončí. (To není samozřejmé a důkaz konečnosti je nutno v obecnosti podávat!) Prozatím jsme však ukazovali algoritmy využívající jen `foreach` cykly, přitom podle naší konvence obsahuje `foreach` cyklus předem danou konečnou množinu hodnot pro řídící proměnnou, neboli náš `foreach` cyklus vždy musí skončit. Ale už v příštím algoritmu využijeme `while` cyklus, u kterého vůbec není jasné *kdy* a *jestli skončí*, a tudíž bude potřebný i důkaz konečnosti.

Metoda 8.7. ***Důkaz konečnosti.***
Máme-li za úkol dokázat, že algoritmus skončí, vhodný postup je následující:

* Sledujeme zvolený celočíselný a zdola ohraničený *parametr algoritmu* (třeba přirozené číslo) a dokážeme, že se jeho hodnota v průběhu algoritmu neustále *ostře zmenšuje*.
* Případně předchozí přístup rozšíříme na zvolenou *k-tici přirozených parametrů* a dokážeme, že se jejich hodnoty v průběhu algoritmu lexikograficky ostře zmenšují.

Pozor, naše „parametry“ vůbec nemusejí být proměnnými v programu.

Algoritmus 8.8. ***Cykly permutace.***
Pro danou permutaci π na n-prvkové neprázdné množině $A = \{1, 2, \ldots, n\}$ vypíšeme její cykly (viz Oddíl 6.4) takto:

```
U ← {1,2,...,n} = A;
while U≠∅ do
    x ← min(U);    (nejmenší prvek množiny)
    začínáme výpis cyklu '⟨';
    while x∈U do
        vytiskneme x;
        U ← U\{x};   x ← π(x);
    done
    ukončíme výpis cyklu '⟩';
done
```

Jak dokážeme správnost tohoto algoritmu? Opět platí, že přímá aplikace indukce podle n nepřinese nic podstatného. Důkaz si tentokrát rozdělíme na dvě části (podle dvou

`while` cyklů). Všimněte se navíc, že tentokrát je nezbytnou součástí důkazu správnosti algoritmu i důkaz, že oba `while` *cykly vždy skončí.*

> **Věta.** Za předpokladu, že vnitřní `while` cyklus pro jakoukoliv počáteční volbu `x` skončí, vypíše cyklus permutace π obsahující x a odebere všechny prvky tohoto cyklu z množiny `U`, Algoritmus 8.8 vždy skončí se správným výsledkem.

`Důkaz`: Postupujeme indukcí podle počtu cyklů v permutaci π. Jediný cyklus v π (báze indukce) je vypsán dle předpokladu věty a množina U zůstane prázdná, tudíž vnější `while` cyklus skončí po první iteraci a výsledek je správný.

Podlě Věty 6.6 se každá permutace dá zapsat jako složení disjunktních cyklů. Nechť π je tedy složena z $\ell > 1$ cyklů. Po první iteraci `while` cyklu zbude v restrikci permutace π na množinu U celkem $\ell - 1$ cyklů. Podle indukčního předpokladu pak tyto zbylé cykly budou správně vypsány a algoritmus skončí. □

`Komentář:` Vidíte, že v tomto důkaze indukcí je indukční krok zcela triviální a důležitý je zde především základ indukce?

> **Věta.** Pokud π je permutace, tak vnitřní `while` cyklus vždy skončí a nalezne v π cyklus obsahující libovolný počáteční prvek $x \in U$. Navíc všechny prvky nalezeného cyklu odebere z množiny U.

`Důkaz`: Zde přímo zopakujeme argument důkazu Věty 6.6: Vezmeme libovolný prvek $x = x_1 \in U$ a iterujeme zobrazení $x_{i+1} = \pi(x_i)$ pro $i = 1, 2 \ldots$, až dojde ke zopakování prvku $x_k = x_j$ pro $k > j \geq 1$. (To musí nastat, neboť A je konečná.) Jelikož prvek x_j byl již odebrán z U, v kroku $x = x_k$ dojde k ukončení našeho `while` cyklu. Nadto je π prostá, a proto nemůže nastat $x_k = x_j = \pi(x_{j-1})$ pro $j > 1$. Takto byl nalezen a odebrán z U cyklus $\langle a_1, \ldots, a_{k-1} \rangle$ a důkaz je hotov. □

8.4 Zajímavé algoritmy aritmetiky

Pro další ukázky důkazových technik pro algoritmy se podíváme na některé méně známé krátké algoritmy z oblasti aritmetiky. Například „zbytkové“ umocňování na velmi vysoké exponenty je podkladem známé RSA šifry:

Algoritmus 8.9. ***Binární postup umocňování.***
Pro daná číslo a, b vypočteme jejich celočíselnou mocninu (omezenou na zbytkové třídy modulo m kvůli prevenci přetečení rozsahu celých čísel v počítači), tj. $c = a^b \mod m$.

```
c ← 1;
while b > 0 do
    if b mod 2 > 0 then c ← (c·a) mod m;
    b ← ⌊b/2⌋;   a ← (a·a) mod m;
done
výsledek c;
```

Zde použijeme k důkazu správnosti algoritmu indukci podle délky ℓ binárního zápisu čísla b.

> **Věta.** Algoritmus 8.9 skončí a vždy správně vypočte hodnotu $c = a^b \mod m$.

`Důkaz`: Báze indukce je pro $\ell = 1$, kdy $b = 0$ nebo $b = 1$. Přitom pro $b = 0$ se cyklus vůbec nevykoná a výsledek je $c = 1$. Pro $b = 1$ se vykoná jen jedna iterace cyklu a výsledek je $c = a \mod m$.

Nechť tvrzení platí pro $\ell_0 \geq 1$ a uvažme $\ell = \ell_0 + 1$. Pak zřejmě $b \geq 2$ a vykonají se alespoň dvě iterace cyklu. Po první iteraci bude $a' = a^2$, $b' = \lfloor b/2 \rfloor$ a $c' = (a^{b \mod 2})$ mod m. Tudíž délka binárního zápisu b' bude jen ℓ_0 a podle indukčního předpokladu zbylé iterace algoritmu skončí s výsledkem

$$c = c' \cdot a'^{b'} \mod m = \left(a^{b \mod 2} \cdot a^{2\lfloor b/2 \rfloor}\right) \mod m = a^b \mod m\,.$$

□

Na závěr lekce si ukážeme jeden netradiční krátký algoritmus a jeho analýzu a důkaz ponecháme zde otevřené. Dokážete popsat, na čem je algoritmus založen?

Algoritmus 8.10. ***Celočíselná odmocnina.***
Pro dané přirozené číslo x vypočteme dolní celou část jeho odmocniny $r = \lfloor \sqrt{x} \rfloor$.

```
p ← x;   r ← 0;
while  p > 0  do
    while  (r + p)^2 ≤ x  do r ← r+p ;
    p ← ⌊p/2⌋ ;
done
výsledek r ;
```

`Poznámka`: Zamysleli jste se, jaký mají algoritmy v tomto oddíle vlastně význam? Vždyť stejné úlohy jistě sami vyřešíte každý jednou jednoduchou `foreach` smyčkou. Podívejte se však (alespoň velmi zhruba) na počet kroků, které zde uvedené algoritmy potřebují vykonat k získání výsledku, a srovnejte si to s počty kroků oněch „jednoduchých“ algoritmů.

Pro skutečně velká vstupní čísla zjistíte propastný rozdíl – s takovým „jednoduchým“ algoritmem, třeba '`foreach i ← 1,...b do c ← c·a mod m done`', se pro obrovské hodnoty b výsledku nikdy nedočkáte, kdežto Algoritmus 8.9 stále poběží velmi rychle. (Spočítáte, jak rychle?) Obdobně je tomu u Algoritmu 8.10.

Rozšiřující studium

Smyslem této lekce bylo především ukázat, že u jednoduchých algoritmů lze (a je vhodné) matematicky dokazovat jejich správnost. Samozřejmě je iluzorní předpokládat, že obdobné důkazy správnosti podáme i pro velké softwarové projekty čítající až milióny řádků, ale postupy a techniky naučené při ověřování jednoduchých algoritmů s úspěchem využijete i při kontrole a ladění jednotlivých částí velkých projektů. Mimoto jsme se na příkladech pokusili ukázat několik zajímavých myšlenek a triků návrhu algoritmů, z nichž zvláště binární umocňování či centrální myšlenka „dynamického“ Algoritmu 8.6 pro tranzitivní uzávěr se opakují v mnoha jiných algoritmech.

O mnoha různých pokročilých technikách tzv. formální verifikace *programů se v případě zájmu dozvíte v pokračujícím studiu. Tyto techniky jsou schopny na vhodném „modelu“ rutinně a matematicky přesně ověřovat (ne)existenci různých běžných programátorských chyb.*

9 Jednoduchý deklarativní jazyk

Úvod

Nyní se vracíme a pokračujeme dále v tématu předchozí lekce, tj. budeme se zabývat matematickým dokazováním vlastností a správnosti algoritmů. Třebaže mnohým mohla přijít už Lekce 8 více než dost formální, není tomu úplně tak; v následujícím si ukážeme (ještě) přesnější přístup založený na myšlenkách funkcionálního programování. Zároveň si tak i procvičíme správné chápání a používání matematických definic.

Cíle

Prvořadě jde o podání matematicky přesné definice jednoduchého deklarativního „programovacího" jazyka pro potřeby formálního dokazování příslušně zapsaných algoritmů. Na tomto základě pak v příští lekci postavíme přehled důkazových technik pro algoritmy.

O „správnosti" programů, podruhé

Vraťme se zpět k fundamentální otázce – jak se máme přesvědčit, že program funguje „správně"?

* V případech, kdy je třeba mít naprostou jistotu správné funkce, je jedinou „dostatečně spolehlivou" možností podat formální matematický důkaz chování algoritmu.
  - Úplné matematické důkazy pro svou složitost dokáží obsáhnout pouze „malé" algoritmy. Přesto je jejich přesné dokazování důležité, neboť právě malé algoritmy obvykle tvoří stavební kameny rozsáhlejších systémů (kde již pak nastupují jiné metody *verifikace*).
* A co tedy důkazy vlastností symbolicky zapsaných (procedurálních) algoritmů z Lekce 8? Všimli jste si, co v nich bylo problematickým bodem?
  - Náš procedurální zápis algoritmu totiž přesně nedefinuje, co je to „*elementární krok*" výpočtu – to je sice většinou docela zřejmé, někdy však může hlavní problém nastat právě zde. Sice by bylo možné použít k definici některý z přesných teoretických modelů výpočtu jako je *Turingův stroj* (nebo třeba i některý vhodný z reálných programovacích jazyků), avšak pak by se formální důkazy staly velmi složitými.
* Vhodnějším řešením (pro potřeby formálního dokazování) se jeví příklon k „*funkcionálnímu*" zápisu algoritmů pomocí matematicky zcela přesných *deklarací*.

9.1 Popis jednoduchého deklarativního jazyka

Z těchto výše popsaných důvodů se nyní soustředíme na podání matematicky zcela přesné definice jednoduchého deklarativního „programovacího" jazyka. Začneme jeho syntaxí (Definice 9.1) a poté přejdeme k jeho sémantice (Definice 9.2) – tedy k formalizaci takto zapsaných pojmů „výpočetního kroku" a „výpočtu".

Definice 9.1. ***Deklarativní programovací jazyk*** (pro přednášky FI: IB000).

* Nechť $Var = \{x, y, z, \dots\}$ je spočetná množina *proměnných*.
* Nechť $Num = \{\mathbf{0}, \mathbf{1}, \dots \mathbf{52}, \dots \mathbf{397}, \dots\}$ je množina všech dekadických zápisů *přirozených* čísel.

∗ Nechť $Fun = \{f, g, h, \dots\}$ je spočetná množina *funkčních symbolů*. Ke každému $f \in Fun$ je přiřazeno číslo $a \in \mathbb{N}$, které nazýváme *arita* f. Dále předpokládáme, že pro každé $a \in \mathbb{N}$ existuje nekonečně mnoho $f \in Fun$ s aritou a.

∗ Množina ***výrazů*** *Exp* je (*induktivně*) definována následující abstraktní syntaktickou rovnicí:

$$\begin{array}{rcl} E & ::= & x \mid \mathbf{n} \\ & \mid & E_1 + E_2 \mid E_1 - E_2 \mid E_1 * E_2 \mid E_1 \div E_2 \mid (\, E_1 \,) \\ & \mid & f(E_1, \cdots, E_a) \\ & \mid & \mathsf{if}\ E_1\ \mathsf{then}\ E_2\ \mathsf{else}\ E_3\ \mathsf{fi} \end{array}$$

V uvedené rovnici je $x \in Var$, $\mathbf{n} \in Num$, $E_i \in Exp$ jsou výrazy, $f \in Fun$ a $a \in \mathbb{N}$ je arita funkce f.

`Poznámka`: Dobře si povšimněte, že ve smyslu Lekce 6 tato induktivní Definice 9.1 není jednoznačná. Neboli jeden výraz, jako třeba $a + 1 - 2 * 3$ lze odvodit několika různými způsoby. Praktickým důsledkem pak je, že není zřejmá „priorita operací“, což lze na druhou stranu ale vždy snadno *napravit přidáním dostatečného počtu závorek*. Tuto abstraktní volnost syntaxe ponecháváme z čistě pragmatického důvodu snadnějšího méně formálního zápisu výrazů. Nakonec bude priorita operací přesně určena následující Definicí 9.2.

`Komentář:` Pro lepší pochopení uvádíme několik příkladů výrazů (*Exp*) z definice.

- $\mathbf{254}$
- $\mathbf{2} + \mathbf{3} * \mathbf{4}$
- $f(\mathbf{2}) \div g(\mathbf{5})$
- $f(\mathbf{2} + x, g(y, \mathbf{3} * y))$
- $\mathsf{if}\ x - \mathbf{1}\ \mathsf{then}\ (\mathbf{2} + f(y))\ \mathsf{else}\ g(x, x)\ \mathsf{fi}$ (čtěme „$\mathsf{if}\ x - 1 \neq 0 \dots$“)

`Definice`: *Deklarace* (v jazyce Definice 9.1) je konečný *systém rovnic* tvaru

$$\begin{array}{rcl} f_1(x_1, \cdots, x_{a_1}) & = & E_{f_1} \\ \vdots & & \vdots \\ f_n(x_1, \cdots, x_{a_n}) & = & E_{f_n} \end{array} ,$$

kde pro každé $1 \leq i \leq n$ platí, že $f_i \in Fun$ je funkce arity a_i, že $x_1, \dots, x_{a_i} \in Var$ jsou proměnné a E_i je výraz, v němž se mohou vyskytovat pouze proměnné $x_1, \dots, x_{a_i}$ a funkční symboly $f_1, \dots, f_n$.

`Komentář:` Opět uvádíme pro osvětení několik příkladů deklarací z naší definice.

- $f(x) = \mathsf{if}\ x\ \mathsf{then}\ x * f(x - \mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi}$
- $\begin{array}{rcl} f(x) & = & g(x - \mathbf{1}, x) \\ g(x, y) & = & \mathsf{if}\ x\ \mathsf{then}\ f(y)\ \mathsf{else}\ \mathbf{3}\ \mathsf{fi} \end{array}$

  Později uvidíme, proč a jak je konvencí našich výpočtů tvrdit $0 - 1$ “$=$” 0.
- $g(x, y) = \mathsf{if}\ x - y\ \mathsf{then}\ x\ \mathsf{else}\ y\ \mathsf{fi}$
- $f(x) = f(x)$ *(Nezapisuje toto náhodou „nekonečnou smyčku“?)*

Deklarace nám udává „soubor pravidel“, podle kterých budeme vyhodnocovat (Definice 9.2) daný výraz. Jinak také lze deklaraci chápat jako zobecnění *rekurentních definic funkcí*.

Neformálně řečeno, deklarace je tedy souborem pravidel pro určení hodnot funkcí s argumenty stojících na levé straně. Hodnoty jsou určeny běžnými aritmetickými výrazy používajícími číselné konstanty, argumenty funkce a hodnoty jiných zde deklarovaných funkcí. Tento poslední aspekt zavádí do deklarací „rekurzivnost“ a dává tak deklaracím jejich „výpočetní sílu“. Nedívejte se na deklarace v žádném případě pohledem procedurálního programování (tedy nejsou to žádná přiřazení hodnot proměnným), lépe na ně nahlédněte pohledem funkcionálního programování.

9.2 Formalizace pojmu „výpočet“

Za výpočet (nad Δ) budeme považovat posloupnost úprav výrazů, které jsou „postaveny“ na naší uvažované deklaraci Δ. To je formálně podchyceno v následujících dvou definicích. Srovnejte si Definici 9.2 také s neformálním pojetím algoritmů jako konečných posloupností elementárních kroků v Oddíle 1.4.

`Definice`: Nechť Δ je deklarace. Symbolem $Exp(\Delta)$ označíme množinu všech výrazů E, které splňují zároveň tyto dvě podmínky:

- E neobsahuje *žádnou proměnnou*.
- Jestliže E obsahuje funkční symbol f, pak f byl v Δ deklarován (tj. na levé straně některé rovnice Δ).

`Komentář:` Dívejte se na množinu $Exp(\Delta)$ jako na soubor „platných vstupů“ (jako u programu) pro deklaraci Δ, nad kterými bude prováděn výpočet.

`Fakt`: Množinu $Exp(\Delta)$ lze podle Definice 9.1 zadat také induktivně:

$$\begin{array}{rcl} E & ::= & \mathbf{n} \mid (E_1) \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 * E_2 \mid E_1 \div E_2 \\ & \mid & f(E_1, \cdots, E_a) \mid \mathsf{if}\ E_1\ \mathsf{then}\ E_2\ \mathsf{else}\ E_3\ \mathsf{fi} \end{array}$$

V uvedené zápise je $\mathbf{n} \in Num$, $f \in Fun$ je funkční symbol deklarovaný v Δ a $a \in \mathbb{N}$ je arita tohoto f.

Jednotlivý krok výpočtu

Definice 9.2. ***Výpočet a krok výpočtu*** v našem deklarativním jazyce.
Relaci „*krok výpočtu*“ $\mapsto \subseteq Exp(\Delta) \times Exp(\Delta)$ definujeme induktivně; místo $(E, F) \in \mapsto$ budeme psát $\boldsymbol{E \mapsto F}$, neboť se pro naše potřeby bude zhusta jednat o funkci.

i) $\mathbf{n} \mapsto \mathbf{n}$ pro každé $\mathbf{n} \in Num$.

ii) Pro $\boldsymbol{E \equiv (E_1)}$ definujeme krok výpočtu takto:

- Jestliže $E_1 \mapsto F \in Num$, pak $(E_1) \mapsto F$.
- Jestliže $E_1 \mapsto F \notin Num$, pak $(E_1) \mapsto (F)$.

iii) Pro $\boldsymbol{E \equiv E_1 + E_2}$ definujeme krok výpočtu takto:

- Jestliže $E_1, E_2 \in Num$, pak $E_1 + E_2 \mapsto \mathbf{z}$, kde $\mathbf{z}$ je dekadický zápis čísla $E_1 + E_2$.
- Jestliže $E_1 \notin Num$ a $E_1 \mapsto F$, pak $E_1 + E_2 \mapsto F + E_2$.

- Jestliže $E_1 \in \mathit{Num}$ a $E_2 \notin \mathit{Num}$ a $E_2 \mapsto F$, pak $E_1 + E_2 \mapsto E_1 + F$.

iv) Pro $\boldsymbol{E \equiv E_1 - E_2}$ definujeme krok výpočtu takto:

- Jestliže $E_1, E_2 \in \mathit{Num}$, pak $E_1 - E_2 \mapsto \mathbf{z}$, kde $\mathbf{z}$ je dekadický zápis čísla $\max\{0, E_1 - E_2\}$. (Pozor na nezápornost výsledku odčítání!)
- Jestliže $E_1 \notin \mathit{Num}$ a $E_1 \mapsto F$, pak $E_1 - E_2 \mapsto F - E_2$.
- Jestliže $E_1 \in \mathit{Num}$ a $E_2 \notin \mathit{Num}$ a $E_2 \mapsto F$, pak $E_1 - E_2 \mapsto E_1 - F$.

v) Pro $\boldsymbol{E \equiv E_1 * E_2}$ definujeme krok výpočtu takto:

- Jestliže $E_1, E_2 \in \mathit{Num}$, pak $E_1 * E_2 \mapsto \mathbf{z}$, kde $\mathbf{z}$ je dekadický zápis čísla $E_1 * E_2$.
- Jestliže $E_1 \notin \mathit{Num}$ a $E_1 \mapsto F$, pak $E_1 * E_2 \mapsto F * E_2$.
- Jestliže $E_1 \in \mathit{Num}$ a $E_2 \notin \mathit{Num}$ a $E_2 \mapsto F$, pak $E_1 * E_2 \mapsto E_1 * F$.

vi) Pro $\boldsymbol{E \equiv E_1 \div E_2}$ definujeme krok výpočtu takto:

- Jestliže $E_1, E_2 \in \mathit{Num}$, pak $E_1 \div E_2 \mapsto \mathbf{z}$, kde $\mathbf{z}$ je dekadický zápis (celé části) čísla $\lfloor E_1/E_2 \rfloor$. Pokud $E_2 \equiv \mathbf{0}$, je $\mathbf{z} \equiv \mathbf{0}$ (dělení nulou!).
- Jestliže $E_1 \notin \mathit{Num}$ a kde $E_1 \mapsto F$, pak $E_1 \div E_2 \mapsto F \div E_2$.
- Jestliže $E_1 \in \mathit{Num}$ a $E_2 \notin \mathit{Num}$ a $E_2 \mapsto F$, pak $E_1 \div E_2 \mapsto E_1 \div F$.

vii) Pro $\boldsymbol{E \equiv}$ if $\boldsymbol{E_1}$ then $\boldsymbol{E_2}$ else $\boldsymbol{E_3}$ fi definujeme krok výpočtu takto:

- Jestliže $E_1 \in \mathit{Num}$ a $E_1 \equiv \mathbf{0}$, pak if E_1 then E_2 else E_3 fi $\mapsto (E_3)$.
- Jestliže $E_1 \in \mathit{Num}$ a $E_1 \not\equiv \mathbf{0}$, pak if E_1 then E_2 else E_3 fi $\mapsto (E_2)$.
- Jestliže $E_1 \notin \mathit{Num}$ a $E_1 \mapsto F$, pak if E_1 then E_2 else E_3 fi $\mapsto$ if F then E_2 else E_3 fi.

viii) Pro $\boldsymbol{E \equiv f(E_1, \cdots, E_k)}$ definujeme krok výpočtu takto:

- Jestliže $E_1, \cdots, E_k \in \mathit{Num}$, pak $f(E_1, \cdots, E_k) \mapsto (E_f(x_1 \restriction E_1, \cdots, x_k \restriction E_k))$. (V tomto formálním zápise se jedná o prosté „textové“ dosazení hodnot E_i za proměnné x_i v deklaraci E_f funkce f v Δ.)
- Jinak $f(E_1, \cdots, E_k) \mapsto f(E_1, \cdots, E_{i-1}, F, E_{i+1}, \cdots, E_k)$, kde i je nejmenší index pro který platí $E_i \notin \mathit{Num}$ a $E_i \mapsto F$.

V zápise jednotlivých bodů vždy platí, že $E_1, E_2, \cdots \in \mathit{Exp}(\Delta)$. Znak $\equiv$ zde znamená „textovou“ rovnost na množině výrazů *Exp*. Při nejednoznačnosti vždy aplikujeme *první použitelné* pravidlo naší definice na *prvním použitelném místě zleva*.

`Definice`: *Reflexivní a tranzitivní uzávěr* relace $\mapsto$ značíme $\mapsto^*$ („*výpočet*“).

Poznámky ke kroku výpočtu

Tato rozsáhlá a důležitá definice si zaslouží několik podstatných připomínek. Za prvé si dobře povšimněte některých „aritmetických“ aspektů výpočtu.

– Výsledek odečítání je vždy nezáporný – všechna záporná čísla jsou nahrazena nulou.

– Výsledek dělení je vždy celočíselný, počítáme jeho dolní celou část.

– *Dělení nulou* je definováno (není chybou), výsledkem je opět nula.

Další připomínka se týká pořadí vyhodnocování ve výrazu — to je přesně dáno pořadím pravidel v Definici 9.2, neboli vždy aplikujeme první pravidlo, které aktuálně lze použít na výraz E, a to na prvním možném místě zleva. Mimo jiné je takto „definována“ nejvyšší priorita vyhodnocení uzávorkovaného výrazu.

Uvědomte si dobře, že definice výpočetního kroku $\mapsto$ je (poněkud skrytě) *rekurzivní*. Třeba krok $(\mathbf{2}*\mathbf{1}) \mapsto \mathbf{2}$ je ve skutečnosti jediným krokem, přestože „vyvolá“ použití dvou pravidel v sobě – vyhodnocení součinu i odstranění závorek.

`Poznámka`: Ještě si uveďme, že naše definice připouští jistý *nedeterminismus* (poznámka jen pro čtenáře, kteří o nedeterminismu už slyšeli): Je možné mít v deklaraci Δ zadaných více rovnic pro tutéž funkci $f()$, pak však $\mapsto$ přestává být funkcí. My se touto možností nebudeme zabývat.

9.3 Příklady výpočtů a důkazů

Po suchém formálním vysvětlení si na příkladech ukážeme, jak dopadá „výpočet“ nad našimi deklaracemi. Definice 9.2 popisuje jediný krok výpočtu, ale kdy má vlastně *celý výpočet* (tj. iterace jednotlivých elementárních kroků podle této definice) skončit? Jak uvidíme i v příkladech, *ukončením výpočtu* budeme rozumnět stav, kdy aktuální výraz je numerickou konstantou (podle Definice 9.2.i už pak k další změně stejně nedojde.)

Příklad 9.3. *Ukážeme si několik ilustrativních „výpočtů“ nad různými deklaracemi. Všimněte si, že při úpravách výrazů si dovolujeme (oproti formální definici) pro zpřehlednění vynechávat zdvojené a vnější závorky.*

- Uvažme deklaraci $f(x) = \mathsf{if}\ x\ \mathsf{then}\ x * f(x-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi}$. Pak třeba $f(\mathbf{3}) \mapsto^* \mathbf{6}$, neboť

$$\begin{array}{llll} f(\mathbf{3}) & \mapsto \mathsf{if}\ \mathbf{3}\ \mathsf{then}\ \mathbf{3}*f(\mathbf{3}-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi} & \mapsto \mathbf{3}*f(\mathbf{3}-\mathbf{1}) & \mapsto \\ \mathbf{3}*f(\mathbf{2}) & \mapsto \mathbf{3}*(\mathsf{if}\ \mathbf{2}\ \mathsf{then}\ \mathbf{2}*f(\mathbf{2}-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi}) & \mapsto \mathbf{3}*(\mathbf{2}*f(\mathbf{2}-\mathbf{1})) & \mapsto \\ \mathbf{3}*(\mathbf{2}*f(\mathbf{1})) & \mapsto \mathbf{3}*(\mathbf{2}*(\mathsf{if}\ \mathbf{1}\ \mathsf{then}\ \mathbf{1}*f(\mathbf{1}-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi})) & \mapsto \mathbf{3}*(\mathbf{2}*(\mathbf{1}*f(\mathbf{1}-\mathbf{1}))) & \mapsto \\ \mathbf{3}*(\mathbf{2}*(\mathbf{1}*f(\mathbf{0}))) & \mapsto \mathbf{3}*(\mathbf{2}*(\mathbf{1}*(\mathsf{if}\ \mathbf{0}\ \mathsf{then}\ \mathbf{0}*f(\mathbf{0}-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi}))) & \mapsto \mathbf{3}*(\mathbf{2}*(\mathbf{1}*\mathbf{1})) & \mapsto \\ \mathbf{3}*(\mathbf{2}*\mathbf{1}) & \mapsto \mathbf{3}*\mathbf{2} & \mapsto \mathbf{6}\,. & \end{array}$$

- Uvažme deklaraci $f(x) = g(x-\mathbf{1}, x)$ a $g(x,y) = \mathsf{if}\ x\ \mathsf{then}\ f(y)\ \mathsf{else}\ \mathbf{3}\ \mathsf{fi}$. Pak například $f(\mathbf{3}) \mapsto^* f(\mathbf{3})$, neboť

$$f(\mathbf{3}) \mapsto g(\mathbf{3}-\mathbf{1}, \mathbf{3}) \mapsto g(\mathbf{2}, \mathbf{3}) \mapsto \mathsf{if}\ \mathbf{2}\ \mathsf{then}\ f(\mathbf{3})\ \mathsf{else}\ \mathbf{3}\ \mathsf{fi} \mapsto f(\mathbf{3})\,.$$

- Uvažme deklaraci $f(x) = f(x)$. Pak pro každé $\mathbf{n} \in \mathit{Num}$ platí

$$f(\mathbf{n}) \mapsto f(\mathbf{n})$$

a podobně $f(f(\mathbf{n})) \mapsto f(f(\mathbf{n}))$. Ale $f(f(\mathbf{2}+\mathbf{3})) \mapsto f(f(\mathbf{5})) \mapsto f(f(\mathbf{5}))$. □

Příklad 9.4. *Jak bude přesně vyhodnocen následující výraz?*

$$1 + 2 - 3 + 4 - 5$$

Řešení: Klíčem k řešení tohoto je správné pochopení Definice 9.2. Prvním použitelným pravidlem našeho deklarativního jazyka je to pro $E \equiv E_1 + E_2$. Je použito

na prvním místě zleva, tj. pro $E_1 \equiv 1$ a $E_2 \equiv 2-3+4-5$. Podle definice je tedy nutno upravit $E_1 \in Num$ i $E_2 \notin Num$, neboli definici aplikovat (rekurzivně) na výraz E_2.

Při vyhodnocování $E_2 = E' \equiv 2-3+4-5$ je prvním použitelným pravidlem opět to pro $E' \equiv E'_1 + E'_2$, přičemž $E'_1 \equiv 2-3$ a $E'_2 \equiv 4-5$. Podle definice je nutno v tomto místě vyhodnotit výraz $E'_1 \mapsto 0$. Celkem v prvním kroce

$$1+2-3+4-5 \;\mapsto\; 1+\;0\;+4-5.$$

Nakonec stejným postupem získáme:

$$1+0+4-5 \mapsto 1+0+0 \mapsto 1+0 \mapsto 1$$

□

Důkaz správnosti programu

Celá naše formalizace deklarativního jazyka směřuje k přesným matematickým důkazům algoritmů, takže si takové hned názorně ukážeme.

Příklad 9.5. *Pro ukázku uvažme deklaraci Δ obsahující pouze rovnici*

$$f(x) = \mathsf{if}\ x\ \mathsf{then}\ x * f(x-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi}\,.$$

Věta. Pro každé $n \in \mathbb{N}$ platí $f(\mathbf{n}) \mapsto^* \mathbf{m}$, kde $\mathbf{m} \equiv n!$.

`Důkaz` povedeme indukcí podle n:

- *Báze* $n = 0$. Platí $f(\mathbf{0}) \mapsto \mathsf{if}\ \mathbf{0}\ \mathsf{then}\ \mathbf{0} * f(\mathbf{0}-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi} \mapsto \mathbf{1}$ a také $0! = 1$.
- *Indukční krok*. Nechť $n+1 \equiv \mathbf{k}$. Pak

$$f(\mathbf{k}) \mapsto \mathsf{if}\ \mathbf{k}\ \mathsf{then}\ \mathbf{k} * f(\mathbf{k}-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi} \mapsto \mathbf{k} * f(\mathbf{k}-\mathbf{1}) \mapsto \mathbf{k} * f(\mathbf{w})\,,$$

kde $\mathbf{w} \equiv k-1 = n$. Podle I.P. platí $f(\mathbf{w}) \mapsto^* \mathbf{u}$, kde $\mathbf{u} \equiv n!$. Proto $\mathbf{k} * f(\mathbf{w}) \mapsto^* \mathbf{k} * \mathbf{u} \mapsto \mathbf{v}$, kde $\mathbf{v} \equiv (n+1) \cdot n! = (n+1)!$. □

`Komentář:` Vidíte, jak „hutný“ a přitom formálně zcela přesný zápis důkazu naše formalizace umožňuje? Promyslete si podrobně všechny jeho kroky ještě jednou a dobře si uvědomte, co z čeho vyplývá a jak na sebe navazují.

Důkazy „neukončenosti“ výpočtů

Jinou častou otázkou je, jak zdůvodnit, že některý výpočet neskončí. K tomu využijeme následující tvrzení navazující na látku o uzávěrech relací:

`Fakt`: Nechť Δ je deklarace. Pro každé $i \in \mathbb{N}$ definujeme relaci $\mapsto^i \subseteq Exp(\Delta) \times Exp(\Delta)$ předpisem $\mapsto^i = \underbrace{\mapsto \circ \cdots \circ \mapsto}_{i}$. Dále definitoricky klademe $\mapsto^0 = \{(E,E) \mid E \in Exp(\Delta)\}$. Pak pro tranzitivní uzávěr (relaci „výpočet“) platí $\mapsto^* = \bigcup_{i=0}^{\infty} \mapsto^i$.

Podle předchozího faktu platí, že $E \mapsto^* F$ právě když $E \mapsto^i F$ pro nějaké $i \in \mathbb{N}$. Navíc musí existovat *nejmenší* i s touto vlastností. Toto pozorování bývá velmi užitečné v důkazech „neukončenosti“ výpočtů.

Příklad 9.6. *Uvažme deklaraci* $f(x) = f(x)$.

Pak pro každé $\mathbf{n} \in \mathit{Num}$ platí, že *neexistuje* žádné $\mathbf{m} \in \mathit{Num}$ takové, že $f(\mathbf{n}) \mapsto^* \mathbf{m}$.

`Důkaz` sporem: Předpokládejme, že existují $\mathbf{n}, \mathbf{m} \in \mathit{Num}$ takové, že $f(\mathbf{n}) \mapsto^* \mathbf{m}$. Pak existuje *nejmenší* $i \in \mathbb{N}$ takové, že $f(\mathbf{n}) \mapsto^i \mathbf{m}$. Jelikož výrazy $f(\mathbf{n})$ a $\mathbf{m}$ jsou různé, platí $i > 0$. Jelikož $\mapsto^i = \mapsto^{i-1} \circ \mapsto$ a $f(\mathbf{n}) \mapsto f(\mathbf{n})$, platí $f(\mathbf{n}) \mapsto^{i-1} \mathbf{m}$, což je spor s minimalitou i. □

Rozšiřující studium

Místo suchého studia si pro lepší seznámení s naším deklarativním jazykem mohou čtenáři pohrát s jeho online interpretem poskytnutým Vaškem Brožkem na `http://arran.fi.muni.cz/ib000/web_vyhodnot.cgi` *nebo s obdobným klientským (javaskriptovým) udělátkem vyvinutým Martinem Jurčou* `http://arran.fi.muni.cz/ib000/js-jurca/`. *Také je velmi vhodné si srovnat zde předloženou látku s předměty vyučujícími tzv. funkcionální programování (na FI třeba IB015). K samotnému pokročilému dokazování vlastností deklarativních algoritmů přistoupíme v příští lekci.*

10 Důkazové postupy pro algoritmy

Úvod

Nakonec si ukážeme, jak formální deklarativní jazyk z Lekce 9 využít k formálně přesným induktivním důkazům vybraných algoritmů. Celý text tak bude sestávat jen a jen z ukázek takových důkazů. Dá se říci, že tato lekce je „vrcholem" v naší snaze o matematické dokazování algoritmů v informatice. Soustřeďte se v ukázkách důkazů na pochopení, jak jednotlivé formální matematické kroky korespondují s průběhem algoritmů.

Cíle

Na podkladu jednotlivých variant důkazů matematickou indukcí z Lekce 2 si ukážeme přehled formálních indukčních důkazových technik aplikovaných na vybrané algoritmy (v zápisech deklarativního jazyka).

10.1 Technika „fixace parametru"

Na první pohled při snaze dokazovat vlastnosti algoritmů indukcí asi budou mít čtenáři největší komplikace tam, kde se v algoritmu vyskytuje najednou více parametrů. S kterým z nich potom můžeme indukci vést? Odpovědi nám poskytnou následující dvě techniky.

První technika důkazu prostě dopředu za některé parametry dosazuje (obecně zvolené) konstanty. Tato technika je vhodná pro případy, kdy je sice v algoritmu více parametrů, ale „zjevně" dochází ke změně jen jednoho (nebo části) z nich a chování algoritmu ke zbylým „neměnným" parametrům je dobře předvídatelné.

Příklad 10.1. *Uvažme deklaraci Δ obsahující pouze rovnici*

$$g(x,y) = \mathsf{if}\ x\ \mathsf{then}\ y + g(x-\mathbf{1},y)\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi}\,.$$

Věta. Pro každé $m,n \in \mathbb{N}$ platí $g(\mathbf{m},\mathbf{n}) \ \mapsto^*\ \mathbf{z}$, kde $\mathbf{z} \equiv m \cdot n$.

Důkaz: Budiž $n \in \mathbb{N}$ libovolné ale pro další úvahy *pevné*. Dokážeme, že pro každé $m \in \mathbb{N}$ platí $g(\mathbf{m},\mathbf{n}) \ \mapsto^*\ \mathbf{z}$, kde $\mathbf{z} \equiv m \cdot n$, indukcí vzhledem k m.

* *Báze* $m = 0$. Platí $g(\mathbf{0},\mathbf{n}) \ \mapsto\ \mathsf{if}\ \mathbf{0}\ \mathsf{then}\ \mathbf{n} + g(\mathbf{0}-\mathbf{1},\mathbf{n})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi} \ \mapsto\ \mathbf{0}$.
* *Indukční krok*. Nechť $m+1 \equiv \mathbf{k}$. Pak

  $$g(\mathbf{k},\mathbf{n}) \ \mapsto\ \mathsf{if}\ \mathbf{k}\ \mathsf{then}\ \mathbf{n} + g(\mathbf{k}-\mathbf{1},\mathbf{n})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi} \ \mapsto\ \mathbf{n} + g(\mathbf{k}-\mathbf{1},\mathbf{n}) \ \mapsto\ \mathbf{n} + g(\mathbf{w},\mathbf{n})\,,$$

  kde je $\mathbf{w} \equiv m$. Podle I.P. platí $g(\mathbf{w},\mathbf{n}) \ \mapsto^*\ \mathbf{u}$ pro $\mathbf{u} \equiv m \cdot n$. Dále $\mathbf{n} + g(\mathbf{w},\mathbf{n}) \ \mapsto^*\ \mathbf{n} + \mathbf{u} \ \mapsto\ \mathbf{v}$, kde $\mathbf{v} \equiv n + (m \cdot n) = (m+1) \cdot n = k \cdot n$, a tím jsme dohromady hotovi s důkazem $g(\mathbf{k},\mathbf{n}) \ \mapsto^*\ \mathbf{v}$. □

10.2 Technika „indukce k součtu parametrů"

Druhou techniku je vhodné použít především v případech, kdy se v průběhu algoritmu vždy některý parametr zmenšuje, ale pokaždé je to některý jiný parametr, takže v indukci se nelze zaměřit jen na jeden z nich.

Příklad 10.2. *Uvažme deklaraci Δ obsahující pouze rovnici*

$$g(x,y) = \mathsf{if}\ x\ \mathsf{then}\ (\mathsf{if}\ y\ \mathsf{then}\ g(x-\mathbf{1},y) + g(x,y-\mathbf{1})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi}\,.$$

Věta. Pro každé $m, n \in \mathbb{N}$ platí $g(\mathbf{m}, \mathbf{n}) \mapsto^* \mathbf{0}$.

Tvrzení této věty přímo nelze dokázat indukcí vzhledem k m, ani indukcí vzhledem k n, neboť u žádného z m, n nemáme zaručeno, že se vždy zmenší. Důkaz lze ovšem postavit na faktu, že se vždy zmenší alespoň jeden z m, n, neboli se vždy zmenší součet m a n. To znamená, že výše uvedené tvrzení nejprve přeformulujeme do následující (matematicky ekvivalentní) podoby:

> **Věta**. Pro každé $i \in \mathbb{N}$ platí, že jestliže $i = m + n$ pro kterákoliv $m, n \in \mathbb{N}$, pak $g(\mathbf{m}, \mathbf{n}) \mapsto^* \mathbf{0}$.

`Důkaz` indukcí vzhledem k i: *Báze* $i = 0$ znamená, že $0 = m + n$ pro $m, n \in \mathbb{N}$, neboli $m = n = 0$. Dokazujeme tedy, že $g(\mathbf{0}, \mathbf{0}) \mapsto^* \mathbf{0}$. Platí

$$g(\mathbf{0}, \mathbf{0}) \mapsto \mathsf{if}\ \mathbf{0}\ \mathsf{then}\ (\mathsf{if}\ \mathbf{0}\ \mathsf{then}\ g(\mathbf{0} - \mathbf{1}, \mathbf{0}) + g(\mathbf{0}, \mathbf{0} - \mathbf{1})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi} \mapsto \mathbf{0}\,.$$

Indukční krok. Nechť $i+1 = m+n$, kde $m, n \in \mathbb{N}$. Nyní rozlišíme tři možnosti (z nichž první dvě jsou svým způsobem jen rozšířeními předchozí báze indukce):

* Pro $m = 0$ platí

$$g(\mathbf{0}, \mathbf{n}) \mapsto \mathsf{if}\ \mathbf{0}\ \mathsf{then}\ (\mathsf{if}\ \mathbf{n}\ \mathsf{then}\ g(\mathbf{0} - \mathbf{1}, \mathbf{n}) + g(\mathbf{0}, \mathbf{n} - \mathbf{1})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi} \mapsto \mathbf{0}\,.$$

* Pro $m > 0, n = 0$ platí

$$g(\mathbf{m}, \mathbf{0}) \mapsto \mathsf{if}\ \mathbf{m}\ \mathsf{then}\ (\mathsf{if}\ \mathbf{0}\ \mathsf{then}\ g(\mathbf{m} - \mathbf{1}, \mathbf{0}) + g(\mathbf{m}, \mathbf{0} - \mathbf{1})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi} \mapsto$$
$$\mapsto \mathsf{if}\ \mathbf{0}\ \mathsf{then}\ g(\mathbf{m} - \mathbf{1}, \mathbf{0}) + g(\mathbf{m}, \mathbf{0} - \mathbf{1})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi} \mapsto \mathbf{0}\,.$$

* Pro $m > 0, n > 0$ platí

$$g(\mathbf{m}, \mathbf{n}) \mapsto \mathsf{if}\ \mathbf{m}\ \mathsf{then}\ (\mathsf{if}\ \mathbf{n}\ \mathsf{then}\ g(\mathbf{m} - \mathbf{1}, \mathbf{n}) + g(\mathbf{m}, \mathbf{n} - \mathbf{1})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi} \mapsto$$
$$\mapsto \mathsf{if}\ \mathbf{n}\ \mathsf{then}\ g(\mathbf{m} - \mathbf{1}, \mathbf{n}) + g(\mathbf{m}, \mathbf{n} - \mathbf{1})\ \mathsf{else}\ \mathbf{0}\ \mathsf{fi} \mapsto g(\mathbf{m} - \mathbf{1}, \mathbf{n}) + g(\mathbf{m}, \mathbf{n} - \mathbf{1})\,.$$

Podle I.P. platí $g(\mathbf{m} - \mathbf{1}, \mathbf{n}) \mapsto^* \mathbf{0}$ a současně $g(\mathbf{m}, \mathbf{n} - \mathbf{1}) \mapsto^* \mathbf{0}$, proto

$$g(\mathbf{m} - \mathbf{1}, \mathbf{n}) + g(\mathbf{m}, \mathbf{n} - \mathbf{1}) \mapsto^* \mathbf{0} + g(\mathbf{m}, \mathbf{n} - \mathbf{1}) \mapsto^* \mathbf{0} + \mathbf{0} \mapsto \mathbf{0}\,.$$

Tím jsme s důkazem matematickou indukcí hotovi. □

Zajímavější verze

Udělejme si předchozí nudný příklad trochu zajímavějším (ale co se týče důkazu stále v podstatě stejným...).

Příklad 10.3. *Uvažme deklaraci Δ obsahující pouze rovnici*

$$g(x, y) = \mathsf{if}\ x\ \mathsf{then}\ (\mathsf{if}\ y\ \mathsf{then}\ g(x - \mathbf{1}, y) + g(x, y - \mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi}\,.$$

Věta. Pro každé $m, n \in \mathbb{N}$ platí $g(\mathbf{m}, \mathbf{n}) \mapsto^* \mathbf{k}$, kde $k = \binom{m+n}{m}$ (kombinační číslo).

Toto tvrzení opět budeme dokazovat indukcí vzhledem k $i = m + n$.

Vzpomeňte si nejprve na známý *Pascalův trojúhelník* kombinačních čísel, který je definovaný rekurentním vztahem

$$\binom{a+1}{b+1} = \binom{a}{b+1} + \binom{a}{b}.$$

Nepřipomíná to trochu naši deklaraci? Je však třeba správně „nastavit“ význam parametrů a, b.

`Důkaz` indukcí vzhledem k i: *Báze* $i = 0$ znamená, že $0 = m + n$ pro $m, n \in \mathbb{N}$, neboli $m = n = 0$. Dokazujeme tedy, že $g(\mathbf{0}, \mathbf{0}) \mapsto^* \mathbf{1}$. Platí

$$g(\mathbf{0},\mathbf{0}) \mapsto \textsf{ if } \mathbf{0} \textsf{ then (if } \mathbf{0} \textsf{ then } g(\mathbf{0}-\mathbf{1},\mathbf{0}) + g(\mathbf{0},\mathbf{0}-\mathbf{1}) \textsf{ else } \mathbf{1} \textsf{ fi) else } \mathbf{1} \textsf{ fi} \mapsto \mathbf{1}.$$

Indukční krok. Nechť $i+1 = m+n$, kde $m, n \in \mathbb{N}$. Opět rozlišíme stejné tři možnosti:

* Pro $m = 0$ platí $\binom{n}{0} = 1$ a

$$g(\mathbf{0},\mathbf{n}) \mapsto \textsf{ if } \mathbf{0} \textsf{ then (if } \mathbf{n} \textsf{ then } g(\mathbf{0}-\mathbf{1},\mathbf{n}) + g(\mathbf{0},\mathbf{n}-\mathbf{1}) \textsf{ else } \mathbf{1} \textsf{ fi) else } \mathbf{1} \textsf{ fi} \mapsto \mathbf{1}.$$

* Pro $m > 0, n = 0$ platí $\binom{m}{m} = 1$ a

$$g(\mathbf{m},\mathbf{0}) \mapsto \textsf{ if } \mathbf{m} \textsf{ then (if } \mathbf{0} \textsf{ then } g(\mathbf{m}-\mathbf{1},\mathbf{0}) + g(\mathbf{m},\mathbf{0}-\mathbf{1}) \textsf{ else } \mathbf{1} \textsf{ fi) else } \mathbf{1} \textsf{ fi} \mapsto$$
$$\mapsto \textsf{ if } \mathbf{0} \textsf{ then } g(\mathbf{m}-\mathbf{1},\mathbf{0}) + g(\mathbf{m},\mathbf{0}-\mathbf{1}) \textsf{ else } \mathbf{1} \textsf{ fi} \mapsto \mathbf{1}.$$

* Pro $m > 0, n > 0$ platí

$$g(\mathbf{m},\mathbf{n}) \mapsto \textsf{ if } \mathbf{m} \textsf{ then (if } \mathbf{n} \textsf{ then } g(\mathbf{m}-\mathbf{1},\mathbf{n}) + g(\mathbf{m},\mathbf{n}-\mathbf{1}) \textsf{ else } \mathbf{1} \textsf{ fi) else } \mathbf{1} \textsf{ fi} \mapsto$$
$$\mapsto \textsf{ if } \mathbf{n} \textsf{ then } g(\mathbf{m}-\mathbf{1},\mathbf{n}) + g(\mathbf{m},\mathbf{n}-\mathbf{1}) \textsf{ else } \mathbf{1} \textsf{ fi} \mapsto g(\mathbf{m}-\mathbf{1},\mathbf{n}) + g(\mathbf{m},\mathbf{n}-\mathbf{1}).$$

Podle I.P. platí $g(\mathbf{m}-\mathbf{1},\mathbf{n}) \mapsto^* \mathbf{k_1}$, kde $\mathbf{k_1} \equiv \binom{m-1+n}{m-1}$, a současně $g(\mathbf{m},\mathbf{n}-\mathbf{1}) \mapsto^* \mathbf{k_2}$, kde $\mathbf{k_2} \equiv \binom{m+n-1}{m}$. Přitom z Pascalova trojúhelníka plyne

$$\binom{m+n-1}{m-1} + \binom{m+n-1}{m} = \binom{(m+n-1)+1}{m} = \binom{m+n}{m},$$

a proto

$$g(\mathbf{m}-\mathbf{1},\mathbf{n}) + g(\mathbf{m},\mathbf{n}-\mathbf{1}) \mapsto^* \mathbf{k_1} + \mathbf{k_2} \mapsto^* \mathbf{k} \equiv \binom{m+n}{m}.$$

□

10.3 Technika „zesílení dokazovaného tvrzení“

Velmi častou situací při dokazování algoritmu je, že se zajímáme o hodnoty některých proměnných nebo „výstupy“ některé funkce, ale ke správnému matematickému důkazu musíme „postihnout“ i chování jiných funkcí a proměnných v algoritmu. Taková situace pak typicky vede na potřebu *zesílení požadovaného tvrzení* v matematické indukci.

Příklad 10.4. *Uvažme deklaraci Δ obsahující tyto rovnice:*

$f(x) =$ if x then $h(x)$ else $\mathbf{1}$ fi
$h(x) =$ if x then $f(x-\mathbf{1}) + h(x-\mathbf{1})$ else $\mathbf{1}$ fi

Věta. Pro každé $n \in \mathbb{N}$ platí $f(\mathbf{n}) \mapsto^* \mathbf{m}$, kde $m = 2^n$.

Požadované tvrzení bohužel nelze přímo dokázat indukcí podle n. Řešením je přeformulování dokazovaného tvrzení do silnější podoby, kterou již indukcí dokázat lze:

Věta. *Pro každé $n \in \mathbb{N}$ platí $f(\mathbf{n}) \mapsto^* \mathbf{m}$ a $h(\mathbf{n}) \mapsto^* \mathbf{m}$, kde $m = 2^n$.*

`Důkaz`, již poměrně snadno indukcí vzhledem k n:

* *Báze* $n = 0$. Platí

$$f(\mathbf{0}) \mapsto \mathsf{if}\ \mathbf{0}\ \mathsf{then}\ h(\mathbf{0})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi} \mapsto \mathbf{1},\ \text{kde } 2^0 = 1,$$
$$h(\mathbf{0}) \mapsto \mathsf{if}\ \mathbf{0}\ \mathsf{then}\ f(\mathbf{0}-\mathbf{1}) + h(\mathbf{0}-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi} \mapsto \mathbf{1}.$$

* *Indukční krok:* Nechť $n + 1 \equiv \mathbf{k}$, pak platí

$$f(\mathbf{k}) \mapsto \mathsf{if}\ \mathbf{k}\ \mathsf{then}\ h(\mathbf{k})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi} \mapsto h(\mathbf{k}) \mapsto$$
$$\mapsto \mathsf{if}\ \mathbf{k}\ \mathsf{then}\ f(\mathbf{k}-\mathbf{1}) + h(\mathbf{k}-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi} \mapsto f(\mathbf{k}-\mathbf{1})+h(\mathbf{k}-\mathbf{1}) \mapsto f(\mathbf{w})+h(\mathbf{k}-\mathbf{1}),$$

kde $\mathbf{w} \equiv k - 1 = n$. Podle I.P. platí $f(\mathbf{w}) \mapsto^* \mathbf{m}$, kde $m = 2^n$. Zároveň také (naše „*zesílení*“) platí i $h(\mathbf{w}) \mapsto^* \mathbf{m}$, a proto

$$f(\mathbf{w}) + h(\mathbf{k}-\mathbf{1}) \mapsto^* \mathbf{m} + h(\mathbf{k}-\mathbf{1}) \mapsto^* \mathbf{m} + h(\mathbf{w}) \mapsto^* \mathbf{m}+\mathbf{m} \mapsto \mathbf{q},$$

kde $q = m + m = 2m = 2 \cdot 2^n = 2^{n+1} = 2^k$. Proto tranzitivně $f(\mathbf{k}) \mapsto \mathbf{q}$ a první část našeho tvrzení platí i pro $n + 1 \equiv \mathbf{k}$.

Podobně je třeba *ještě dokončit* druhou část tvrzení.

$$h(\mathbf{k}) \mapsto \mathsf{if}\ \mathbf{k}\ \mathsf{then}\ f(\mathbf{k}-\mathbf{1}) + h(\mathbf{k}-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi} \mapsto$$
$$f(\mathbf{k}-\mathbf{1}) + h(\mathbf{k}-\mathbf{1}) \mapsto^* f(\mathbf{w}) + h(\mathbf{k}-\mathbf{1}),$$

kde $\mathbf{w} \equiv k - 1 = n$. Podle I.P. platí $f(\mathbf{w}) \mapsto^* \mathbf{m}$, kde $m = 2^n$, a také $h(\mathbf{w}) \mapsto^* \mathbf{m}$, tudíž opět

$$f(\mathbf{w}) + h(\mathbf{k}-\mathbf{1}) \mapsto^* \mathbf{m} + h(\mathbf{k}-\mathbf{1}) \mapsto^* \mathbf{m} + h(\mathbf{w}) \mapsto^* \mathbf{m}+\mathbf{m} \mapsto \mathbf{q},$$

kde $q = m + m = 2 \cdot 2^n = 2^{n+1} = 2^k$. Proto $h(\mathbf{k}) \mapsto \mathbf{q}$ a i druhá část našeho tvrzení platí pro $n + 1 \equiv \mathbf{k}$. □

10.4 Dva dobře známé školní algoritmy

Následuje příklad, který zajisté každý, kdo přičichl k programování, už v nějaké podobě pozná. My si jej uvádíme především z výukových důvodů.

Číslice dekadického zápisu

Příklad 10.5. *Mějme přirozené číslo x. Jednotlivé číslice i-tého řádu jeho dekadického zápisu získáme deklarací*

$$c(x, i) = \mathsf{if}\ i\ \mathsf{then}\ c(x \div \mathbf{10}, i-\mathbf{1})\ \mathsf{else}\ x - \mathbf{10} * (x \div \mathbf{10})\ \mathsf{fi}.$$

Dokažte:

Věta. Pro každá $m, i \in \mathbb{N}$ platí $c(\mathbf{m}, \mathbf{i}) \mapsto^* \mathbf{s}$, kde $\mathbf{s}$ je číslice řádu i (počítáno od nultého zprava) v dekadickém zápise čísla $\mathbf{m}$, nebo $\mathbf{s} \equiv \mathbf{0}$ v případě, že dekadický zápis čísla $\mathbf{m}$ má méně než $i+1$ číslic.

Důkaz: Použijeme techniku fixace parametru m při indukci podle i.

* *Báze* $i = 0$. Platí

$$c(\mathbf{m}, \mathbf{0}) \mapsto^* \mathbf{m} - \mathbf{10} * (\mathbf{m} \div \mathbf{10}) \mapsto^* \mathbf{t}, \text{ kde } t = m \mod 10\,.$$

V tomto výsledku mod znamená známou funkci modulo definovanou vztahem $x = \lfloor x/y \rfloor \cdot y + (x \mod y)$. Tudíž t je poslední číslicí dekadického zápisu m.

* *Indukční krok:* Nechť $i + 1 \equiv \mathbf{j}$, pak platí

$$c(\mathbf{m}, \mathbf{j}) \mapsto \mathsf{if}\ \mathbf{j}\ \mathsf{then}\ c(\mathbf{m} \div \mathbf{10}, \mathbf{j} - \mathbf{1})\ \mathsf{else}\ \mathbf{m} - \mathbf{10} * (\mathbf{m} \div \mathbf{10})\ \mathsf{fi} \mapsto$$
$$\mapsto c(\mathbf{m} \div \mathbf{10}, \mathbf{j} - \mathbf{1}) \mapsto^* c(\mathbf{p}, \mathbf{w}),$$

kde $\mathbf{p} \equiv \lfloor m/10 \rfloor$ a $\mathbf{w} \equiv j - 1 = i$. Podle I.P. platí $c(\mathbf{p}, \mathbf{w}) \mapsto^* \mathbf{t}$ a t je číslice i-tého řádu dekadického zápisu čísla p. Jelikož $m = 10p + (m \mod 10)$ a násobení deseti v dekadické soustavě znamená posun číslic v zápise „o jedno doleva“, je t zároveň číslice $i + 1 = j$-tého řádu dekadického zápisu čísla m. To je přesně co bylo třeba dokázat.□

Euklidův algoritmus

Již z dávných dob antiky pochází následující zajímavý a koneckonců velmi jednoduchý algoritmus teorie čísel. (Víte, pro zajímavost, jestli se tehdy vůbec mluvilo o algoritmech a programech, nebo jako to bylo?)

Věta 10.6. *Uvažme deklaraci* Δ *obsahující pouze rovnici*

$$g(x, y) = \mathsf{if}\ x - y\ \mathsf{then}\ g(x - y, y)\ \mathsf{else}\ (\mathsf{if}\ y - x\ \mathsf{then}\ g(x, y - x)\ \mathsf{else}\ x\ \mathsf{fi})\ \mathsf{fi}\,.$$

Pak pro každé <u>*nenulové*</u> $m, n \in \mathbb{N}$ *platí* $g(\mathbf{m}, \mathbf{n}) \mapsto^* \mathbf{z}$*, kde* z *je největší společný dělitel čísel* m, n*.*

Důkaz indukcí k $i = m + n$. (Tj. dokazujeme následující tvrzení: *Pro každé* $i \geq 2$ *platí, že jestliže* $i \geq m + n$*, kde* $m, n \in \mathbb{N}$*,* $m, n > 0$*, pak* z *je největší společný dělitel čísel* m, n*.*)

V *bázi* pro $i = 2$ je $m, n = 1$ a platí

$$g(\mathbf{1}, \mathbf{1}) \mapsto \mathsf{if}\ \mathbf{1} - \mathbf{1}\ \mathsf{then}\ g(\mathbf{1} - \mathbf{1}, \mathbf{1})\ \mathsf{else}\ (\mathsf{if}\ \mathbf{1} - \mathbf{1}\ \mathsf{then}\ g(\mathbf{1}, \mathbf{1} - \mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi})\ \mathsf{fi} \mapsto$$
$$\mapsto \mathsf{if}\ \mathbf{0}\ \mathsf{then}\ g(\mathbf{1} - \mathbf{1}, \mathbf{1})\ \mathsf{else}\ (\mathsf{if}\ \mathbf{1} - \mathbf{1}\ \mathsf{then}\ g(\mathbf{1}, \mathbf{1} - \mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi})\ \mathsf{fi} \mapsto$$
$$\mapsto \mathsf{if}\ \mathbf{1} - \mathbf{1}\ \mathsf{then}\ g(\mathbf{1}, \mathbf{1} - \mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi} \mapsto \mathsf{if}\ \mathbf{0}\ \mathsf{then}\ g(\mathbf{1}, \mathbf{1} - \mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi} \mapsto \mathbf{1}\,.$$

Indukční krok. Nechť $i + 1 = m + n$ kde $m, n \in \mathbb{N}$. Probereme tři možnosti:

* $m = n$. Pak

$$g(\mathbf{m}, \mathbf{n}) \mapsto \mathsf{if}\ \mathbf{m} - \mathbf{n}\ \mathsf{then}\ g(\mathbf{m} - \mathbf{n}, \mathbf{n})\ \mathsf{else}\ (\mathsf{if}\ \mathbf{n} - \mathbf{m}\ \mathsf{then}\ g(\mathbf{m}, \mathbf{n} - \mathbf{m})\ \mathsf{else}\ \mathbf{m}\ \mathsf{fi})\ \mathsf{fi} \mapsto$$
$$\mathsf{if}\ \mathbf{0}\ \mathsf{then}\ g(\mathbf{m} - \mathbf{n}, \mathbf{n})\ \mathsf{else}\ (\mathsf{if}\ \mathbf{n} - \mathbf{m}\ \mathsf{then}\ g(\mathbf{m}, \mathbf{n} - \mathbf{m})\ \mathsf{else}\ \mathbf{m}\ \mathsf{fi})\ \mathsf{fi} \mapsto$$
$$\mathsf{if}\ \mathbf{n} - \mathbf{m}\ \mathsf{then}\ g(\mathbf{m}, \mathbf{n} - \mathbf{m})\ \mathsf{else}\ \mathbf{m}\ \mathsf{fi} \mapsto \mathsf{if}\ \mathbf{0}\ \mathsf{then}\ g(\mathbf{m}, \mathbf{n} - \mathbf{m})\ \mathsf{else}\ \mathbf{m}\ \mathsf{fi} \mapsto \mathbf{m}\,.$$

∗ $m < n$. Pak

$$g(\mathbf{m},\mathbf{n}) \;\mapsto\; \mathsf{if}\ \mathbf{m}-\mathbf{n}\ \mathsf{then}\ g(\mathbf{m}-\mathbf{n},\mathbf{n})\ \mathsf{else}\ (\mathsf{if}\ \mathbf{n}-\mathbf{m}\ \mathsf{then}\ g(\mathbf{m},\mathbf{n}-\mathbf{m})\ \mathsf{else}\ \mathbf{m}\ \mathsf{fi})\ \mathsf{fi} \;\mapsto$$
$$\mathsf{if}\ \mathbf{0}\ \mathsf{then}\ g(\mathbf{m}-\mathbf{n},\mathbf{n})\ \mathsf{else}\ (\mathsf{if}\ \mathbf{n}-\mathbf{m}\ \mathsf{then}\ g(\mathbf{m},\mathbf{n}-\mathbf{m})\ \mathsf{else}\ \mathbf{m}\ \mathsf{fi})\ \mathsf{fi} \;\mapsto$$
$$\mathsf{if}\ \mathbf{n}-\mathbf{m}\ \mathsf{then}\ g(\mathbf{m},\mathbf{n}-\mathbf{m})\ \mathsf{else}\ \mathbf{m}\ \mathsf{fi} \;\mapsto\; \mathsf{if}\ \mathbf{z}\ \mathsf{then}\ g(\mathbf{m},\mathbf{n}-\mathbf{m})\ \mathsf{else}\ \mathbf{m}\ \mathsf{fi} \;\mapsto$$
$$g(\mathbf{m},\mathbf{n}-\mathbf{m}) \;\mapsto\; g(\mathbf{m},\mathbf{k})\,,$$

kde $\mathbf{k} \equiv n-m$. Platí $m+k = m+(n-m) = n \leq i$, takže podle I.P. také platí $g(\mathbf{m},\mathbf{k}) \mapsto^* \mathbf{z}$, kde z je největší společný dělitel čísel m a $n-m$. Ověříme, že z je největší společný dělitel čísel m a n.

- Jelikož číslo z dělí čísla m a $n-m$, dělí i jejich součet $(n-m)+m = n$. Celkem z je společným dělitelem m a n.
- Buď d nějaký společný dělitel čísel m a n. Pak d dělí také rozdíl $n-m$. Tedy d je společný dělitel čísel m a $n-m$. Jelikož z je největší společný dělitel čísel m a $n-m$, nutně d dělí z a závěr platí.

∗ $m > n$. Pak

$$g(\mathbf{m},\mathbf{n}) \;\mapsto^*\; g(\mathbf{m}-\mathbf{n},\mathbf{n}) \;\mapsto\; g(\mathbf{k},\mathbf{n})\,,$$

kde $\mathbf{k} \equiv m-n$. Podle I.P. platí $g(\mathbf{k},\mathbf{n}) \mapsto^* \mathbf{z}$, kde z je největší společný dělitel čísel $m-n$ a n. Podobně jako výše ověříme, že z je také největší společný dělitel čísel m a n. □

`Poznámka`: Jak byste výše uvedený zápis Euklidova algoritmu vylepšili, aby správně „počítal“ největšího společného dělitele i v případech, že $m = 0$ nebo $n = 0$?
Co v takových případech selže při současném zápise?

11 Nekonečné množiny a Zastavení algoritmu

Úvod

Bystrého čtenáře může snadno napadnout myšlenka, proč se vlastně zabýváme dokazováním správnosti algoritmů a programů, když by to přece (snad?) mohl za nás dělat automaticky počítač samotný. Bohužel to však nejde a je hlavním cílem této lekce ukázat matematické důvody proč tomu tak je.

Konkrétně si dokážeme, že nelze algoritmicky rozhodnout, zda se daný algoritmus na svém vstupu zastaví nebo ne. Hlavními nástroji, které použijeme, budou nekonečné množiny a důkazová technika tzv. Cantorovy diagonály, která se ve velké míře používá právě v teoretické informatice. Touto lekcí se tak krátce vymaníme ze zajetí naivní teorie množin, která právě v této oblasti má své nepřekonatelné limity. (Pro zvídavé – obdobně, ale mnohem složitěji, lze dokázat že ani matematické důkazy nelze obecně algoritmicky konstruovat...)

Cíle

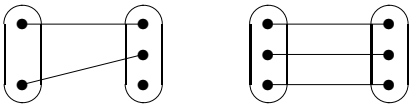
Zavedeme si ve zjednodušeném (a stále „poněkud naivním“) pohledu nekonečné množiny a na nich techniku důkazu Cantorovou diagonálou. Pak tuto klíčovou důkazovou techniku teoretické informatiky využijeme k důkazu algoritmické neřešitelnosti problému zastavení.

11.1 O nekonečných množinách a kardinalitě

Zatímco v naivní teorii jsme si velikost konečné množiny definovali jednoduše jako *počet jejích prvků* vyjádřený přirozeným číslem, pro nekonečné množiny je situace obtížná a mnohem méně intuitivní. Co nám třeba brání zavést pro velikost nekonečné množiny *symbol* ∞? Ponejvíce závažný fakt, že není „nekonečno“ jako „nekonečno“ (Věta 11.2)!

Právě proto si pro určení mohutnosti nekonečných množin musíme vypomoci následujícím příměrem: Velikosti dvou hromádek jablek dokážeme i bez počítání porovnat tak, že budeme z obou hromádek po řadě odebírat dvojice jablek (z každé jedno), až dokud první hromádka nezůstane prázdná – druhá hromádka pak je větší (nebo nejvýše rovna) té první.

`Definice`: Množina A je „*nejvýše tak velká*“ jako množina B, právě když existuje injektivní funkce $f : A \to B$. Množiny A a B jsou „*stejně velké*“ právě když mezi nimi existuje bijekce. V případech nekonečných množin pak místo “velikosti” mluvíme formálně o jejich *kardinalitě*.



`Komentář:` Uvedená definice kardinality množin „funguje“ velmi dobře i pro nekonečné množiny:

* Například $\mathbb{N}$ a $\mathbb{Z}$ mají stejnou kardinalitu (jsou „stejně velké“, tzv. *spočetně nekonečné*).
* Lze snadno ukázat, že i $\mathbb{Q}$ je spočetně nekonečná, tj. existuje bijekce $f : \mathbb{N} \to \mathbb{Q}$.
* Existují ale i nekonečné množiny, které jsou „striktně větší“ než libovolná spočetná množina (příkladem je $\mathbb{R}$ ve Větě 11.2).
* Později dokážeme, že existuje nekonečná posloupnost nekonečných množin, z nichž *každá je striktně větší než všechny předchozí*.

Pro porovnávání velikostí množin někdy s výhodou využijeme následující přirozené, ale nelehké tvrzení (bez důkazu):

Věta 11.1. *Pro libovolné dvě množiny A, B (i nekonečné) platí, že pokud existuje injekce $A \to B$ a zároveň i injekce $B \to A$, pak existuje bijekce mezi A a B.*

Cantorova diagonála, aneb kolik je reálných čísel

Prvním klíčovým poznatkem ukazujícím na neintuitivní chování nekonečných množin je následující důkaz, který dal historicky vzniknout metodě tzv. *Cantorovy diagonály.*

Věta 11.2. *Neexistuje žádné surjektivní zobrazení $g : \mathbb{N} \to \mathbb{R}$.*

Důsledek 11.3. *Neexistuje žádné injektivní (tudíž ani bijektivní) zobrazení $h : \mathbb{R} \to \mathbb{N}$. Neformálně řečeno, reálných čísel je striktně více než všech přirozených.*

`Důkaz` (Věty 11.2 sporem): Nechť takové g existuje a pro zjednodušení se omezme jen na funkční hodnoty v intervalu $(0, 1)$. Podle hodnot zobrazení g si takto můžeme „uspořádat“ dekadické zápisy všech reálných čísel v intervalu $(0, 1)$ po řádcích do tabulky:

$$\begin{array}{lllllllllllll} g(0) = & 0. & \mathbf{1} & 5 & 4 & 2 & 7 & 5 & 7 & 8 & 3 & 2 & 5 & \dots \\ g(1) = & 0. & & \mathbf{4} & & & & & & & & & & \dots \\ g(2) = & 0. & & & \mathbf{1} & & & & & & & & & \dots \\ g(3) = & 0. & & & & \mathbf{3} & & & & & & & & \dots \\ g(4) = & 0. & & & & & \mathbf{9} & & & & & & & \dots \\ \vdots & \vdots & & & & & & \ddots & & & & & & \end{array}$$

Nyní sestrojíme číslo $\alpha \in (0, 1)$ následovně; jeho i-tá číslice za desetinnou čárkou bude 1, pokud v i-tém řádku tabulky na diagonále není 1, jinak to bude 2. V našem příkladě $\alpha = 0.21211\ldots$

Kde se naše číslo α v tabulce nachází? (Nezapomeňme, g byla surjektivní, takže tam α musí být!) Kostrukce však ukazuje, že α se od každého čísla v tabulce liší na aspoň jednom desetinném místě, to je spor. *(Až na drobný technický detail s rozvojem $\ldots \bar{9}$.)* □

11.2 „Naivní“ množinové paradoxy

Analogickým způsobem k Větě 11.2 lze dokázat následovné zobecnění.

Věta 11.4. *Nechť M je libovolná množina. Pak existuje injektivní zobrazení $f : M \to 2^M$, ale neexistuje žádné bijektivní zobrazení $g : M \to 2^M$.*

`Důkaz`: Dokážeme nejprve existenci f. Stačí ale položit $f(x) = \{x\}$ pro každé $x \in M$. Pak $f : M \to 2^M$ je zjevně injektivní.

Neexistenci g dokážeme sporem. Předpokládejme tedy naopak, že existuje bijekce $g : M \to 2^M$. Uvažme množinu $K \subseteq M$ definovanou takto:

$$K = \{x \in M \mid x \notin g(x)\}.$$

Jelikož g je bijektivní a $K \in 2^M$, musí existovat $y \in M$ takové, že $g(y) = K$. Nyní rozlišíme dvě možnosti:

– $y \in g(y)$. Tj. $y \in K$. Pak ale $y \notin g(y)$ z definice K, spor.

– $y \notin g(y)$. To podle definice K znamená, že $y \in K$, tj. $y \in g(y)$, spor. □

`Komentář:` Dvě navazující poznámky.

- Technika použitá v důkazech *Vět 11.2* a *11.4* se nazývá *Cantorova diagonální metoda*, nebo také zkráceně *diagonalizace*.

  Konstrukci množiny K lze znázornit pomocí následující tabulky:

	a	b	c	d	$\cdots$
$g(a)$	$\checkmark$	$-$	$-$	$\surd$	$\cdots$
$g(b)$	$\surd$	$-$	$-$	$\surd$	$\cdots$
$g(c)$	$-$	$\surd$	$-$	$\surd$	$\cdots$
$g(d)$	$-$	$-$	$\surd$	$\checkmark$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

  Symbol $\surd$ resp. $-$ říká, že prvek uvedený v záhlaví sloupce patří resp. nepatří do množiny uvedené v záhlaví řádku. Tedy např. $d \in g(b)$ a $a \notin g(d)$. Množina K poté obsahuje ty diagonální prvky označené $-$, tj. „převrací“ význam diagonály.

- Z toho, že nekonečna mohou být „různě velká“, lze lehce odvodit řadu dalších faktů. V jistém smyslu je např. množina všech problémů větší než množina všech algoritmů (obě množiny jsou nekonečné), a proto nutně existují problémy, které *nejsou algoritmicky řešitelné*.

Cantorův paradox

Naivní teorie množin, jak jsme si ji uvedli i v tomto předmětu, trpí mnoha neduhy a nepřesnostmi, které vyplynou na povrch především při „neopatrné manipulaci“ s nekonečnými množinami. Abychom se těmto „neopatrnostem“ vyhnuli bez přílišné formalizace, dva základní z těchto paradoxů si nyní ukážeme.

Příklad 11.5. *Uvážíme-li nyní nekonečnou posloupnost množin*

$$A_1, A_2, A_3, A_4, \cdots$$

kde $A_1 = \mathbb{N}$ *a* $A_{i+1} = 2^{A_i}$ *pro každé* $i \in \mathbb{N}$, *je vidět, že všechny množiny jsou nekonečné a každá je striktně větší než libovolná předchozí.*

Kde však v tomto řazení mohutností bude „*množina všech množin*“? Na tuto otázku, jak sami asi cítíte, nelze podat odpověď. Co to však znamená? □

∗ Takto se koncem 19. století objevil první *Cantorův paradox* nově vznikající teorie množin.

∗ Dnešní moderní vysvětlení paradoxu je jednoduché, prostě „množinu všech množin“ nelze definovat, taková v matematice neexistuje.

Brzy se však ukázalo, že je ještě *mnohem hůř*...

Russelův paradox

`Fakt`: Není pravda, že *každý soubor prvků* lze považovat za množinu.

* Nechť $X = \{M \mid M \text{ je množina taková, že } M \notin M\}$. Platí $X \in X$?
  - Ano. Tj. $X \in X$. Pak ale X splňuje $X \notin X$.
  - Ne. Pak X splňuje vlastnost $X \notin X$, tedy X je prvkem X, tj., $X \in X$.
* Obě možné odpovědi vedou ke sporu. X tedy nelze prohlásit za množinu. Jak je ale něco takového vůbec možné?

`Komentář:` Vidíte u Russelova paradoxu podobnost přístupu s Cantorovou diagonalizací? Russelův paradox se objevil začátkem 20. století a jeho „jednoduchost“ zasahující úplné základy matematiky (teorie množin) si vynutila hledání seriózního rozřešení a rozvoj formální matematické logiky.

Pro ilustraci tohoto paradoxu, slyšeli jste už, že „*v malém městečku žije holič, který holí právě ty muže městečka, kteří se sami neholí*“? Zmíněné paradoxy naivní teorie množin zatím v tomto kurzu nerozřešíme, ale zapamatujeme si, že většina matematických a informatických disciplín vystačí s „*intuitivně bezpečnými*“ množinami.

11.3 Algoritmická neřešitelnost problému zastavení

Výše vysvětlené myšlenky diagonalizace a principů základních paradoxů naivní teorie množin sice vypadají „velmi matematicky“. Přesto je však téměř beze změny lze aplikovat i na bytostně informatickou otázku, zda lze algoritmicky poznat, pro které vstupy se daný algoritmus vůbec zastaví. Negativní odpověď na tuto otázku je jedním z fundamentálních výsledků informatiky a přitom má překvapivě krátký a čistý důkaz diagonalizací.

`Fakt`: Uvědomme si (velmi neformálně) několik základních myšlenek.

* Program v každém programovacím jazyce je konečná posloupnost složená z *konečně mnoha* symbolů (písmena, číslice, mezery, speciální znaky, apod.) Nechť Σ je množina všech těchto symbolů. Množina všech programů je tedy jistě podmnožinou množiny $\bigcup_{i \in \mathbb{N}} \Sigma^i$, která je spočetně nekonečná. Existuje tedy bijekce f mezi množinou $\mathbb{N}$ a množinou všech programů. Pro každé $i \in \mathbb{N}$ označme symbolem P_i program $f(i)$. Pro *každý program* P tedy existuje $j \in \mathbb{N}$ takové, že $P = P_j$.
* Každý možný vstup každého možného programu lze zapsat jako *konečnou posloupnost* symbolů z konečné mnočiny Γ. Množina všech možných vstupů je tedy spočetně nekonečná a existuje bijekce g mezi množinou $\mathbb{N}$ a množinou všech vstupů. Pro každé $i \in \mathbb{N}$ označme symbolem V_i vstup $g(i)$.
* Předpokládejme, že *existuje program Halt*, který pro dané $i, j \in \mathbb{N}$ zastaví s výstupem *ano*/*ne* podle toho, zda P_i pro vstup V_j zastaví, nebo ne.
* Tento předpoklad dále dovedeme ke *sporu* dokazujícímu, že problém zastavení nemá algoritmické řešení.

Věta 11.6. *Neexistuje program Halt, který by pro vstup (P_i, V_j) správně rozhodl, zda se program P_i zastaví na vstupu V_j.*

`Důkaz`: Sporem uvažme program *Diag* s následujícím kódem:

```
input k;
if  Halt(k,k) = ano  then  while true do ; done
```

(Program *Diag*(k) má na rozdíl od *Halt* jen jeden vstup k, což bude důležité.)

Fungování programu *Diag* lze znázornit za pomocí následující tabulky:

	P_0	P_1	P_2	P_3	$\cdots$
V_0	$\checkmark$	$-$	$-$	$\surd$	$\cdots$
V_1	$\surd$	$-$	$-$	$\surd$	$\cdots$
V_2	$-$	$\surd$	$-$	$\surd$	$\cdots$
V_3	$-$	$-$	$\surd$	$\checkmark$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Symbol $\surd$ resp. $-$ říká, že program uvedený v záhlaví sloupce zastaví resp. nezastaví pro vstup uvedený v záhlaví řádku. Program *Diag* „*zneguje*“ diagonálu této tabulky.

Podle našeho předpokladu (*Diag* je program a posloupnost P_i zahrnuje *všechny programy*) existuje $j \in \mathbb{N}$ takové, že $Diag = P_j$. Zastaví *Diag* pro vstup V_j?

– *Ano*. Podle kódu *Diag* pak ale tento program vstoupí do nekonečné smyčky, tedy nezastaví.

– *Ne*. Podle kódu *Diag* pak ale `if` test neuspěje, a tento program tedy zastaví.

Předpoklad existence programu *Halt* tedy vede ke sporu. □

`Komentář:` Otázkami algoritmické (ne)řešitelnosti problémů se zabývá *teorie vyčíslitelnosti*. Metoda diagonalizace se také často využívá v *teorii složitosti* k důkazu toho, že dané dvě složitostní třídy jsou různé.

Rozšiřující studium

Látka této lekce zabrousila až do teoretických hlubin matematické logiky a teorie množin. Další studium v těchto oblastech se dá očekávat hlavně u studentů specificky zaměřených teoretickým směrem (a mířících spíše do akademické než aplikační sféry), zajímajících se o matematiku samotnou nebo o teorii vyčíslitelnosti. Proto také uvedené pokročilé poznatky nebudou vyžadovány u zkoušky tohoto předmětu.

12 Délka výpočtu algoritmu

Úvod

Mimo samotné správnosti výsledku vypočteného zapsaným algoritmem je ještě jedno neméně důležité hledisko k posouzení vhodnosti algoritmu k řešení zadané úlohy. Jedná se o čas, který algoritmus stráví výpočtem.

Asi netřeba argumentovat, že přehnaně dlouhá doba odezvy programu je každému uživateli nepříjemná. A co třeba v real-time systémech, kde si zdržení prostě nemůžeme dovolit. Obligátní odpověď „kupme si rychlejší počítač" bohužel není vždy řešením, jak při pokročilém studiu složitosti algoritmů sami poznáte. Mnohem větší potenciál zrychlení se skrývá v algoritmech samotných a jejich efektivním návrhu.

Cíle

V této lekci definujeme délku výpočtu algoritmu, přičemž definici postavíme na deklarativním jazyce z Lekce 9. Poté si ukážeme, jak se matematicky popisuje asymptotické chování funkcí, což se využívá především pro zjednodušené studium složitosti výpočtu algoritmu.

12.1 O významu délky výpočtu algoritmu

Uvažme deklarativní jazyk Definice 9.1.

`Definice`: Délkou výpočtu výrazu F nad deklarací Δ rozumíme nejmenší přirozené k takové, že pro něj existuje $\mathbf{m} \in \mathit{Num}$ pro něž $F \mapsto^k \mathbf{m}$. (Když takové $\mathbf{m}$ neexistuje, klademe $k = \infty$.)

`Komentář:` Jaká je délka výpočtu následujících výrazů?

- $\mathbf{3+4-5*6}$; tři kroky $\mathbf{3+4-5*6} \mapsto \mathbf{3+4-30} \mapsto \mathbf{3+0} \mapsto \mathbf{3}$.
- $\mathbf{3+(5-4)*(6\div 2)}$; tentokrát čtyři kroky $\mathbf{3+(5-4)*(6\div 2)} \mapsto \mathbf{3+1*(6\div 2)} \mapsto \mathbf{3+1*3} \mapsto \mathbf{3+3} \mapsto \mathbf{6}$.
- $\mathbf{2011}$; žádný krok, tj. $k = 0$.

Příklad 12.1. *Pro ukázku uvažme deklaraci Δ obsahující pouze rovnici*

$$f(x) = \mathsf{if}\ x\ \mathsf{then}\ x * f(x-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi}\,.$$

Věta. Pro každé $n \in \mathbb{N}$ je délka výpočtu výrazu $f(\mathbf{n})$ rovna $4n+2$.

`Důkaz` povedeme indukcí podle n:

- *Báze* $n = 0$. Platí $f(\mathbf{0}) \mapsto \mathsf{if}\ \mathbf{0}\ \mathsf{then}\ \mathbf{0} * f(\mathbf{0}-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi} \mapsto \mathbf{1}$, což jsou přesně 2 kroky, tj. $4 \cdot 0 + 2$.
- *Indukční krok.* Nechť $n+1 \equiv \mathbf{k}$. Pak

$$f(\mathbf{k}) \mapsto \mathsf{if}\ \mathbf{k}\ \mathsf{then}\ \mathbf{k} * f(\mathbf{k}-\mathbf{1})\ \mathsf{else}\ \mathbf{1}\ \mathsf{fi} \mapsto \mathbf{k} * f(\mathbf{k}-\mathbf{1}) \mapsto \mathbf{k} * f(\mathbf{w})\,,$$

kde $\mathbf{w} \equiv k-1 = n$. To jsou přesně 3 kroky. Podle I.P. je délka výpočtu výrazu $f(\mathbf{w})$ rovna $4n+2$. Poté následuje ještě jeden poslední krok vynásobení $\mathbf{k}$. Celkem se provedlo $3+4n+2+1 = 4(n+1)+2 = 4k+2$ kroků. □

Počítat přesně nebo raději ne?

`Komentář:` Jaký má smysl určení přesného počtu kroků algoritmu při dnešních CPU? Copak jsme dnes schopni jednoznačně říci, jak dlouho jedna instrukce CPU trvá? Z druh strany, i když víme, že algoritmus A třeba potřebuje $2n$ kroků výpočtu a algoritmus B třeba potřebuje $3n$ kroků, je mezi nimi až takový rozdíl? Stačí, když B spustíme na dvakrát rychlejším počítači a poměr se hned obrátí. Obě tyto prakticky motivované úvahy nás povedou k poznání, že aditivní a multiplikativní faktory funkce počtu kroků algoritmu jsou vlastně zanedbatelné.

12.2 Asymptotické značení a odhady funkcí

Zajímá-li nás jen rychlost růstu funkce $f(n)$ v závislosti na n, zaměřujeme se především na tzv. *asymptotické chování* f při velkých hodnotách n. V popisu f nás tedy nezajímají ani různé přičtené "drobné členy", které se významněji projevují jen pro malá n, ani konstanty, kterými je f násobena a které jen ovlivňují číselnou hodnotu $f(n)$, ale ne rychlost růstu.

`Komentář:` Tak například funkce $f(n) = n^2$ roste (zhruba) stejně rychle jako $f'(n) = 100000000n^2$ i jako $f''(n) = 0.00000001n^2 - 100000000n - 1000000$. Naopak $h(n) = 0.00000000001n^3$ roste mnohem rychleji než $f'(n) = 100000000n^2$.

Pro porovnávání rychlostí růstů funkcí nám slouží následující terminologie.

`Definice`: Nechť $g : \mathbb{R} \to \mathbb{R}$ je daná funkce. Pro funkci $f : \mathbb{R} \to \mathbb{R}$ píšeme

$$f \in O\big(g\big)$$

pokud existují konstanty $A, B > 0$ takové, že

$$\forall n \in \mathbb{R}^+ : f(n) \leq A \cdot g(n) + B\,.$$

V praxi se obvykle (i když matematicky méně přesně) píše místo $f \in O\big(g\big)$ výraz

$$f(n) = O\big(g(n)\big)\,.$$

Znamená to, slovně řečeno, že funkce f *neroste rychleji* než funkce g. (I když pro malá n třeba může být $f(n)$ mnohem větší než $g(n)$.)

`Poznámka`: Kromě vlastnosti $f \in O(g)$ se někdy setkáte i s vlastností $f \in o(g)$, která znamená $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ (funkce f *roste striktně pomaleji* než g).

`Definice`: Píšeme $f \in \Omega\big(g\big)$, neboli $f(n) = \Omega\big(g(n)\big)$, pokud $g \in O(f)$. Dále píšeme $f \in \Theta\big(g\big)$, neboli $f(n) = \Theta\big(g(n)\big)$, pokud $f \in O(g)$ a zároveň $f \in \Omega(g)$, neboli $g \in O(f)$.

Výraz $f(n) = \Theta\big(g(n)\big)$ pak čteme jako "funkce f *roste stejně rychle* jako funkce g".

`Značení`: O funkci $f(n)$ říkáme:

* $f(n) = \Theta(n)$ je *lineární* funkce,
* $f(n) = \Theta(n^2)$ je *kvadratická* funkce,
* $f(n) = \Theta(\log n)$ je *logaritmická* funkce,

* $f(n) = O(n^c)$ pro nějaké $c > 0$ je *polynomiální* funkce,
* $f(n) = \Omega(c^n)$ pro nějaké $c > 1$ je *exponenciální* funkce.

Příklad 12.2. *Příklady růstů různých funkcí.*

Funkce $\boldsymbol{f(n) = \Theta(n)}$: pokud n vzroste na dvojnásobek, tak hodnota $f(n)$ taktéž vzroste (zhruba) na dvojnásobek. To platí jak pro funkci $f(n) = n$, tak i pro $1000000000n$ nebo $n + \sqrt{n}$, atd.

Funkce $\boldsymbol{f(n) = \Theta(n^2)}$: pokud n vzroste na dvojnásobek, tak hodnota $f(n)$ vzroste (zhruba) na čtyřnásobek. To platí jak pro funkci $f(n) = n^2$, tak i pro $1000n^2 + 1000n$ nebo $n^2 - 99999999n - 99999999$, atd.

Naopak pro funkci $\boldsymbol{f(n) = \Theta(2^n)}$: pokud n vzroste byť jen o 1, tak hodnota $f(n)$ už vzroste (zhruba) na dvojnásobek. To je obrovský rozdíl exponenciálních proti polynomiálním funkcím.

Pokud vám třeba funkce $\boldsymbol{999999n^2}$ připadá velká, jak stojí ve srovnání s 2^n? Zvolme třeba $n = 1000$, kdy $999999n^2 = 999999000000$ je ještě rozumně zapsatelné číslo, ale $2^{1000} \simeq 10^{300}$ byste už na řádek nenapsali. Pro $n = 10000$ je rozdíl ještě mnohem výraznější! □

Příklad 12.3. *(opakovaný) Zjistěte, kolik znaků* `'x'` *v závislosti na celočíselné hodnotě* n *vstupního parametru* `n` *vypíše následující algoritmus.*

Algoritmus 12.4.

```
foreach  i ← 1,2,3,...,n-1,n  do
    foreach  j ← 1,2,3,...,i-1,i  do
        vytiskni  'x';
    done
done
```

Zatímco v Lekci 8 jsme trochu zdlouhavě indukcí dokazovali, že výsledkem je $\frac{1}{2}n(n+1)$ `'x'`, nyní si mnohem snadněji odvodíme, že počet `'x'` je $\Theta(n^2)$, což je dostačující asymptotická odpověď ve většině informatických aplikací.

`Důkaz`: Shora hned odhadneme, že každá z n iterací vnějšího cyklu vytiskne po $i \leq n$ znaků `'x'`, takže celkem je nejvýše n^2 `'x'`. Naopak zdola hned vidíme, že posledních $n/2$ iterací vnějšího cyklu vytiskne $i \geq n/2$ znaků `'x'`, takže celkem je alespoň $(n/2)\cdot(n/2) = n^2/4$ `'x'`. Z toho podle definice hned vyjde asymptotický odhad $\Theta(n^2)$. □

12.3 Rekurentní odhady

V tomto oddíle si uvedeme krátký přehled některých rekurentních vzorců, se kterými se můžete setkat při řešení časové složitosti (převážně rekurzivních) algoritmů.

Lema 12.5. *Nechť* $a_1, \ldots, a_k, c > 0$ *jsou kladné konstanty takové, že* $a_1 + \cdots + a_k < 1$, *a funkce* $T : \mathbb{N} \to \mathbb{N}$ *splňuje nerovnost*

$$T(n) \leq T(\lceil a_1 n \rceil) + T(\lceil a_2 n \rceil) + \cdots + T(\lceil a_k n \rceil) + cn\,.$$

Pak $T(n) = O(n)$.

Důkaz: Zvolme $\varepsilon > 0$ takové, že $a_1 + \cdots + a_k < 1 - 2\varepsilon$. Pak pro dostatečně velká n platí (i se zaokrouhlením nahoru) $\lceil a_1 n \rceil + \cdots + \lceil a_k n \rceil \leq (1-\varepsilon)n$, řekněme pro všechna $n \geq n_0$. Dále zvolme dostatečně velké $d > 0$ tak, že $\varepsilon d > c$ a zároveň $d > \max\left\{\frac{1}{n}T(n) : n = 1, \ldots, n_0\right\}$.

Dále už snadno indukcí podle n dokážeme $T(n) \leq dn$ pro všechna $n \geq 1$:

- Pro $n \leq n_0$ je $T(n) \leq dn$ podle naší volby d.
- Předpokládejme, že $T(n) \leq dn$ platí pro všechna $n < n_1$, kde $n_1 > n_0$ je libovolné. Nyní dokážeme i pro n_1

$$T(n_1) \leq T(\lceil a_1 n_1 \rceil) + \cdots + T(\lceil a_k n_1 \rceil) + cn_1 \leq$$
$$\leq d \cdot \lceil a_1 n_1 \rceil + \cdots + d \cdot \lceil a_k n_1 \rceil + cn_1 \leq$$
$$\leq d \cdot (1-\varepsilon)n_1 + cn_1 \leq dn_1 - (\varepsilon d - c)n_1 \leq dn_1 .$$

□

Lema 12.6. *Nechť $k \geq 2$ a $a_1, \ldots, a_k, c > 0$ jsou kladné konstanty takové, že $a_1 + \cdots + a_k = 1$, a funkce $T : \mathbb{N} \to \mathbb{N}$ splňuje nerovnost*

$$T(n) \leq T(\lceil a_1 n \rceil) + T(\lceil a_2 n \rceil) + \cdots + T(\lceil a_k n \rceil) + cn . \tag{1}$$

Pak $T(n) = O(n \cdot \log n)$.

Důkaz (poněkud nematematický náznak): Bylo by možno postupovat obdobně jako v předchozím důkaze, ale výpočty by byly složitější. Místo formálního důkazu indukcí nyní předestřeme poměrně jednoduchou úvahu zdůvodňující řešení $T(n) = O(n \cdot \log n)$.

Představme si, že upravujeme pravou stranu výrazu (1) v následujících krocích: V každém kroku rozepíšeme každý člen $T(m)$ s dostatečně velkým argumentem m rekurzivní aplikací výrazu (1) (s $T(m)$ na levé straně). Jelikož $a_1 + \cdots + a_k = 1$, součet hodnot argumentů všech $T(\cdot)$ ve zpracovávaném výrazu bude stále zhruba n. Navíc po zhruba $t = \Theta(\log n)$ krocích už budou hodnoty argumentů všech $T(\cdot)$ "malé" (nebude dále co rozepisovat), neboť $0 < a_i < 1$ a tudíž $a_i^t \cdot n < 1$ pro všechna i. Při každém z kroků našeho rozpisu se ve výrazu (1) přičte hodnota $cn = O(n)$, takže po t krocích bude výsledná hodnota

$$T(n) = t \cdot O(n) + O(n) = O(n \cdot \log n) .$$

Vyzkoušejte si tento postup sami na konkrétním příkladě $T'(n) \leq 2T'\left(\frac{n}{2}\right) + n$. □

V obecnosti je známo:

Lema 12.7. *Nechť $a \geq 1$, $b > 1$ jsou konstanty, $f : \mathbb{N} \to \mathbb{N}$ je funkce a pro funkci $T : \mathbb{N} \to \mathbb{N}$ platí rekurentní vztah*

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + f(n) .$$

Pak platí:

- *Je-li $f(n) = O(n^c)$ a $c < \log_b a$, pak $T(n) = O(n^{\log_b a})$.*
- *Je-li $f(n) = \Theta(n^{\log_b a})$, pak $T(n) = O(n^{\log_b a} \cdot \log n)$.*
- *Je-li $f(n) = \Theta(n^c)$ a $c > \log_b a$, pak $T(n) = O(n^c)$.*

Důkaz tohoto obecného tvrzení přesahuje rozsah našeho předmětu. Všimněte si, že nikde ve výše uvedených řešeních nevystupují počáteční podmínky, tj. hodnoty $T(0), T(1), T(2), \ldots$ – ty jsou "skryté" v naší $O()$-notaci. Dále v zápise pro zjednodušení zanedbáváme i necelé části argumentů, které mohou být zaokrouhlené.

Příklad 12.8. *Algoritmus merge-sort pro třídění čísel pracuje zhruba následovně:*

- *Danou posloupnost n čísel rozdělí na dvě (skoro) poloviny.*
- *Každou polovinu setřídí zvlášť za použití rekurentní aplikace merge-sort.*
- *Tyto dvě už setříděné poloviny "slije" (anglicky merge) do jedné setříděné výsledné posloupnosti.*

Jaký je celkový počet jeho kroků?

Nechť na vstupu je n čísel. Při rozdělení na poloviny nám vzniknou podproblémy o velikostech $\lceil n/2 \rceil$ a $\lfloor n/2 \rfloor$ (pozor na necelé poloviny). Pokud počet kroků výpočtu označíme $T(n)$, pak rekurzivní volání trvají celkem

$$T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)\,.$$

Dále potřebujeme $c \cdot n$ kroků (kde c je vhodná konstanta) na slití obou částí do výsledného setříděného pole. Celkem tedy vyjde

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + cn \leq T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil) + cn$$

a to už je tvar řešený v Lematu 12.6 pro $a_1 = a_2 = \frac{1}{2}$. Výsledek tedy je $T(n) = O(n \cdot \log n)$.
(Stejný výsledek by bylo možno získat i z Lematu 12.7 pro $a = b = 2$.) □

Příklad 12.9. *Předpokládejme na vstupu dvě „dlouhá" $2m$-místná čísla (pro jednoduchost v dekadickém zápise) A a B. Součin $A \cdot B$ můžeme rychleji (než školním postupem) vypočítat takto:*

- *Čísla si „rozdělíme na poloviny", tj. na čtyři m-místná čísla taková, že $A = A_1 + 10^m A_2$ a $B = B_1 + 10^m B_2$.*
- *Rekurzivní aplikací téhož postupu vypočteme tři součiny $C_1 = A_1 \cdot B_1$, $C_2 = A_2 \cdot B_2$ a $D = (A_1 + A_2) \cdot (B_1 + B_2)$.*
- *Dále už jen sečteme (ověřte si snadno sami)*

$$A \cdot B = C_1 + 10^m (D - C_1 - C_2) + 10^{2m} C_2\,.$$

Jaký je celkový počet jeho kroků?

Postupujeme obdobně předchozímu příkladu. Nechť $T(n)$ je počet kroků výpočtu pro vstupní čísla A a B délky nejvýše $n = 2m$. Tři rekurzivní volání pro součiny se aplikují na čísla délek nejvýše m a $m+1$ (pro $(A_1 + A_2) \cdot (B_1 + B_2)$), takže trvají celkem $T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil + 1)$. Pomocné výpočtu součtu a rozdílu snadno vykonáme v $O(n)$ krocích a pomocné násobení mocninou 10 je pouhým posunutím číslic. Celkem tedy

$$T(n) = 2T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil + 1) + O(n) \leq 3T(\lceil n/2 \rceil + 1) + O(n)\,.$$

Navíc člen „+1“ lze v argumentu asymptoticky zanedbat, a proto podle Lematu 12.7 vypočítáme $\log_2 3 \simeq 1.585 > 1$ a $T(n) = O(n^{\log_2 3}) \simeq O(n^{1.585})$, což je výrazně rychlejší než školní postup v čase $O(n^2)$ pro velká n. □

Rozšiřující studium

Poslední lekce našeho učiva nás dovedla do zajímavé a užitečné oblasti výpočetní složitosti. S problematikou zjišťování délky výpočtu („rychlosti“) algoritmu se studenti určitě setkají v předmětech zabývajících se návrhem algoritmů. Alespoň základní znalost problematiky složitosti by měl mít každý programátor, aby dokázal psát rozumně efektivní programy.

Na otázky rychlosti algoritmů pokročile navazuje teorie výpočetní složitosti, která se dále zabývá otázkami, proč pro mnohé problémy efektivní algoritmy neexistují (a existovat nejspíše ani nemohou). Později se tak studenti setkají třeba s pojmem NP-úplnosti – základním kritériem „efektivní neřešitelnosti“ problému.

Závěrem

Gratulujeme všem, kteří se naším nelehkým učebním textem „prokousali“ až sem, a přejeme mnoho úspěchů v dalším studiu informatiky.

Konečně znovu připomínáme, že nedílnou součástí našeho studijního textu je Interaktivní osnova předmětu IB000 v IS MU a v ní přiložené online odpovědníky určené k procvičování přednesené látky.

`http://is.muni.cz/el/1433/podzim2011/IB000/index.qwarp`